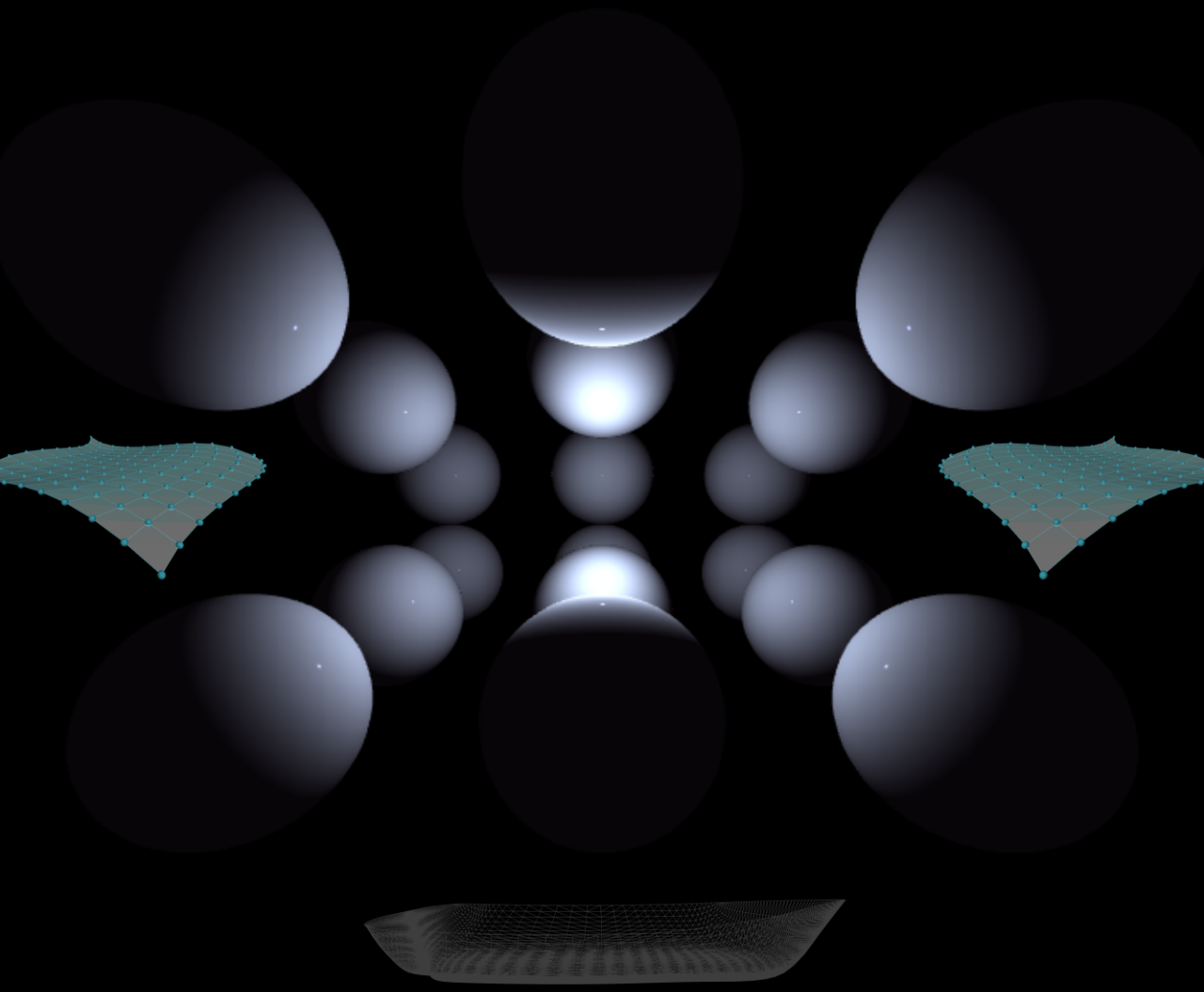
Introducción a la Graficación por Computadora



Melissa Méndez Servín

RED educativa digital escartes

Introducción a la graficación por computadora

Melissa Méndez Servín
Universidad Nacional Autónoma de México



Fondo editorial RED Descartes

Córdoba (España) 2022

Título de la obra:
Introducción a la graficación por computadora

Autora:
Melissa Méndez Servín

Código JavaScript para el libro: Joel Espinosa Longi, IMATE, UNAM.
Fuentes: Lora, Cormorant Garamond, Noto Sans SC, Caveat, UbuntuMono, Glacial Indifference y Press Start 2P.
Fórmulas matemáticas: KaTeX

Red Educativa Digital Descartes
Córdoba (España)
descartes@proyectodescartes.org
https://proyectodescartes.org

Proyecto iCartesiLibri
https://proyectodescartes.org/iCartesiLibri/index.htm

ISBN: 978-84-18834-29-5

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Tabla de contenido

Introducción **7**

1. Introducción a la graficación por computadora **9**

1.1 Aplicaciones de la graficación por computadora 9

1.2 Cámara estenopeica (pinhole camera) 13

1.3 El Sistema de Visión Humano 19

1.4 Color 22

1.4.1 Modelo RGB 23

1.4.2 Modelo HSV 24

1.4.3 Espacio de color y gamut 26

1.5 El pipeline gráfico 28

1.5.1 Aplicación 29

1.5.2 Procesamiento Geométrico (o de vértices) 29

1.5.3 Rasterización 34

1.5.4 Procesamiento de pixeles (o de fragmentos) 35

1.6 Referencias bibliográficas 37

2. Conceptos preliminares de álgebra lineal **41**

2.1 Ángulos, grados y radianes 41

2.2 Un poco de trigonometría 43

2.3 Vectores 44

2.3.1 Multiplicación de vectores por un escalar 46

2.3.2 Suma de vectores 47

2.3.3 Resta de vectores 48

2.3.4 Espacios vectoriales, bases y coordenadas 49

2.3.5 Producto punto 56

2.3.6 Producto cruz 61

2.4 Matrices 64

2.4.1 Operaciones de las matrices 65

2.4.2 Sistemas de ecuaciones lineales 70
2.4.3 Determinantes 74
2.4.4 Cambio de base 79
2.5 Líneas y Planos 81
2.5.1 Líneas 81
2.5.2 Planos 84
2.6 Referencias bibliográficas 87
3. Modelado geométrico 89
3.1 Vértices, aristas y caras 89
3.2 Mallas poligonales 90
3.2.1 Tira y abanico de triángulos 95
3.2.2 Orientación 96
3.3 Curvas de Bézier 97
3.3.1 Parches de Bezier 105
3.4 Referencias bibliográficas 110
4. Transformaciones geométricas en 2D y 3D 113
4.1 Transformaciones geométricas en 2D 113
4.1.1 Escalamiento 114
4.1.2 Rotación 115
4.1.3 Traslación 117
4.2 Transformaciones en coordenadas homogéneas 118
4.3 Transformaciones geométricas en 3D 122
4.4 Transformación de la normal 128
4.5 Referencias bibliográficas 130
5. Transformaciones de vista en 3D 133
5.1 Transformación de la cámara 133
5.2 Transformación de proyección 136
5.2.1 Transformación de proyección ortográfica 137
5.2.2 Transformación de proyección de perspectiva 139

5.3 Transformación de viewport 144
5.4 Referencias bibliográficas 146
6. Modelos de iluminación y sombreado 147
6.1 Reflexión ambiental 152
6.2 Reflexión difusa (superficies de Lambert) 153
6.3 Reflexión especular 157
6.4 Modelo de iluminación de Phong 161
6.5 Modelo de iluminación Blinn-Phong 163
6.6 Materiales 164
6.7 Fuentes de luz 167
6.7.1 Luz puntual 167
6.7.2 Luz direccional 169
6.7.3 Luz de reflector 171
6.8 Transparencia 174
6.9 Referencias bibliográficas 178
7. Texturas 181
7.1 Pipiline de texturas 182
7.1.1 Función de proyección 183
7.1.2 Función de correspondencia 185
7.1.3 Valores de la textura 187
7.1.4 Texturas en el pipeline gráfico 188
7.2 Texturas 2D o de imágenes 188
7.3 Mapeo de materiales 198
7.4 Mapeo de normales y espacio tangente 199
7.5 Mapeo de desplazamiento 204
7.6 Texturas procedurales 206
7.7 Mapeo ambiental (environment mapping) 209
7.8 Referencias bibliográficas 214

Introducción

El objetivo de este libro es proporcionar un material de apoyo para aquellos que estén interesados en el área de graficación por computadora. Se presentan temas introductorios a este ámbito de conocimiento, en particular del renderizado en tiempo real, complementados con ejemplos interactivos hechos con Webgl 2 e imágenes.

El término de Graficación por computadora involucra todo aquello relacionado con la creación y manipulación de imágenes en una computadora, para ser más específicos, la graficación por computadora es el campo que estudia todos los algoritmos, métodos y técnicas para la generación de imágenes sintéticas dada una colección de datos de entrada. La entrada puede ser la descripción de una escena en 3D de un videojuego, o bien, datos de algún experimento científico para representación del mismo, entre muchas otras. *REF*. [1] [2] [7].

Si bien, la graficación por computadora requiere de manera inevitable el conocimiento de ciertas APIs, hardware, y formatos de archivos específicos, este libro no tiene como cometido el enseñar a utilizar un API en algún lenguaje, software y/o hardware en particular, pues lo que se busca es dar un enfoque teórico de lo que son los fundamentos de la graficación, animando a los interesados a no limitarse y complementar lo aprendido realizando sus propias implementaciones en el ambiente de trabajo que más deseen.

Como nota personal me gustaría hacer un especial agradecimiento a mi madre, a mi padre, a Emmanuel, a Diego y, sobre todo, a mi profesor Joel Espinosa Longi, por el gran e invaluable apoyo que me brindaron durante todo el desarrollo de este libro, aportándome siempre muy buenas sugerencias, conocimientos y ánimo.

CAPÍTULO I

Introducción a la graficación por computadora

1.1 Aplicaciones de la graficación por computadora

La *graficación por computadora* (GC) es un campo multidisciplinario bastante amplio, donde tanto computólogos, matemáticos, físicos, ingenieros, artistas y otros practicantes comparten un mismo objetivo "mostrar un mundo a través de una ventana", nos podemos referir como mundo a un modelo digital, una simulación, o bien, cualquier representación visual que se busque mostrar, y como ventana a cualquier medio para mostrar imágenes, como un proyector, la pantalla de un monitor, tablet, entre otros. Por lo que nos podemos dar una idea de las multiples aplicaciones que se pueden obtener en este campo sobre los diferentes ámbitos, veamos algunas de estas aplicaciones:

Industria del entretenimiento

Creación de películas o caricaturas sintéticas, publicidad, efectos visuales y videojuegos.

Ingeniería Mecánica

Diseño de prototipos virtuales de partes mecánicas para su construcción, utilizando sistemas CAD/CAM (*Computer-Aided Design/ Computer-Aided Manufacturing*).

Arquitectura

Uso del software CAD para la creación de planos de alguna estructura arquitectónica, visualizaciones de espacios antes y después de una construcción planeada.

Diseño

Diseño y creación de productos, haciendo uso de sistemas CAD/CAM, promoviendo la creatividad del diseñador al permitirle experimentar con varias formas antes de producir la idea final.

Patrimonio cultural

Reconstrucciones virtuales de templos, monumentos, piezas antiguas, o bien, reconstrucciones hipotéticas de escenas.

Medicina

Simulaciones virtuales de cirugías para entrenamiento y visualización de datos dados por algún instrumento de diagnóstico.

Asimismo, se han ido desarrollando un gran número de áreas de especialización, las cuales han ido evolucionando de acuerdo a las necesidades de los usuarios y al avance tecnológico en el hardware y software. Las principales suelen ser las siguientes:

Renderización

Es el proceso para generar una imagen final dada una entrada ordenada de datos, se suele clasificar dependiendo del algoritmo que se use, generalmente se toma como referencia el tiempo que a éste le tome en producir una imagen sintética (*Real time rendering*, *Offline rendering*), o bien, el tipo de técnica de renderizado que utilice (*Photorealistic rendering*, *non-photorealistic rendering* o *information visualization*).

Modelado

Es la especificación matemática del mundo a representar, es decir, se describen los objetos y sus propiedades de un modo que éstos puedan ser almacenados en la computadora, por ejemplo, una manzana puede ser descrita como un conjunto de puntos 3D, los cuales forman caras ordenadas, con un modelo de iluminación especifico para describir cómo interactúa la luz con la manzana.

Animación

Es la creación de una secuencia de imágenes con una computadora, que tiene como finalidad producir la ilusión de movimiento. Si bien, hace uso de las dos áreas anteriores, no se encarga del estudio de éstas, pues su objetivo es añadir alguna animación sobre un rango de tiempo específico a los modelos, con la cual modifica algún aspecto de éstos, como su color, posición, apariencia, entre otros.

Y entre algunas otras áreas de relevancia se encuentran:

Simulación

Es el proceso en el que un modelo matemático es ejecutado y representado con una computadora, la cual es usada para simular un comportamiento y su repercusión en el mundo real, o bien, dentro de un sistema físico impuesto, permitiendo revisar la fiabilidad del modelo, por lo cual es útil para modelar sistemas en física, astrofísica, climatología, química, biología, economía, ingeniería, etc. Por ejemplo, el movimiento de las partículas del agua en una ola, o bien, la simulación en tiempo real de un escenario de entrenamiento, como volar un avión. Las simulaciones garantizan un menor costo y mayor seguridad para los usuarios.

Fotografía Computacional

Es el uso de las técnicas de graficación por computadora, visión artificial y proceso digital de imágenes empleadas para mejorar el potencial de la fotografía digital y la calidad de la captura digital de imágenes, desarrollando otras formas de capturar objetos, escenas y ambientes. Permitiéndonos producir equipos fotográficos de bajo costo capaces de identificar caras, hacer reenfoques, panorámicas, e incluso hacer que el resultado luzca similar o mejor al de un equipo caro.

Escaneo en 3D

Es el proceso de colectar información de objetos del mundo real para crear un modelo 3D, desarrollando muchos dispositivos y algoritmos para obtener la geometría y apariencia del objeto real, dichos modelos son utilizados para crear visualizaciones.

Proceso Digital de Imágenes

Es la aplicación de técnicas y algoritmos para la manipulación de imágenes 2D, para obtener información de éstas, o bien, mejorar su calidad.

Realidad Virtual

Es la simulación de un entorno virtual en 3D generado por computadora, que intenta hacer experimentar al usuario una sensación de estar inmerso en él. Aunque puede ser proyectado simplemente en la pantalla de un dispositivo como un monitor, se suelen utilizar otras tecnologías tales como cascos de realidad virtual, guantes, o incluso diseñar espacios, para intensificar las sensación de realidad y que además le permita una mejor interacción con el mundo virtual, creando imágenes realistas, sonidos y otras sensaciones.

Realidad Aumentada

Es la visualización de un espacio real en donde a los objetos tangibles de éste se les añade información gráfica generada por computadora, es decir, elementos virtuales son agregados al mundo real, cubriendo total o parcialmente a objetos del entorno, y permitiéndole al usuario interactuar en tiempo real dentro de este espacio mixto.

1.2 Cámara estenopeica (*pinhole camera*)

Para producir imágenes por computadora de escenas 3D, es importante entender primero cómo son producidas las imágenes de manera tradicional, con una cámara física y en el sistema de visión humano (que es el siguiente subtema), pues también vivimos en un mundo tridimensional.

La cámara estenopeica o cámara *pinhole* es la cámara real más simple que existe, lo cual la hace el modelo más utilizado en GC, ya que es más sencillo de reproducir y menos costoso. Es importante entender el funcionamiento de esta cámara ya que retomaremos algunos conceptos de ésta más adelante.

La *cámara estenopeica* es muy simple, consta de una caja a prueba de luz, con un agujero o apertura muy pequeña en el centro de uno de sus lados, y una película fotográfica o algún material fotosensible en la cara interior contraria al agujero, sobre la cual se proyecta la imagen invertida de la escena exterior cuando la apertura es abierta (Véase la **Figura 1.1**).

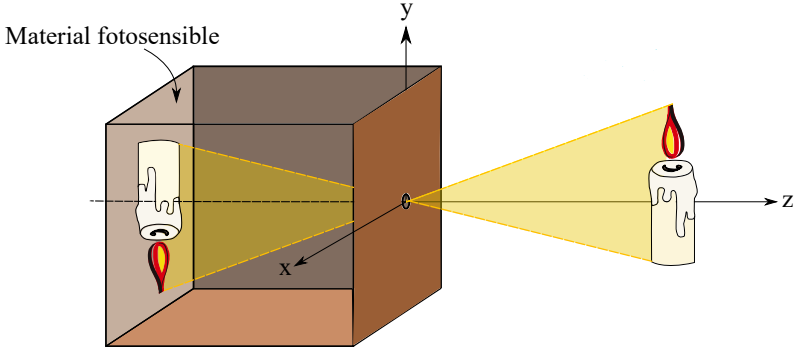


Figura 1.1. Cámara pinhole.

El principio que sigue esta cámara es el mismo que el de una cámara obscura, fenómeno que fue descubierto desde la antigüedad y que permitía estudiar eventos como el eclipse solar. Dicho fenómeno se experimentaba dentro cuartos obscuros con una pequeña abertura, la cual permitía la entrada de la luz, y formaba una imagen invertida de la escena exterior. El siguiente video es de *National Geographic* (todos los derechos reservados) y muestran cómo convertir tú propio cuarto en una cámara obscura.



En el mundo real tenemos varias fuentes de luz, siendo el sol la principal de éstas. Y cuando los rayos de luz chocan contra un objeto, éste puede absorberlos y/o reflejarlos en todas las direcciones. Por lo que, cuando tomamos una foto, solo aquellos rayos de luz que son reflejados del exterior hacia la cámara pasan a través de la pequeña apertura para formar la imagen sobre la película.

Supongamos que colocamos nuestra cámara a lo largo del eje z, con la apertura en el centro del sistema de coordenadas, mirando hacia $z+$. Y que la película fotográfica se encuentra a una distancia d de la apertura.

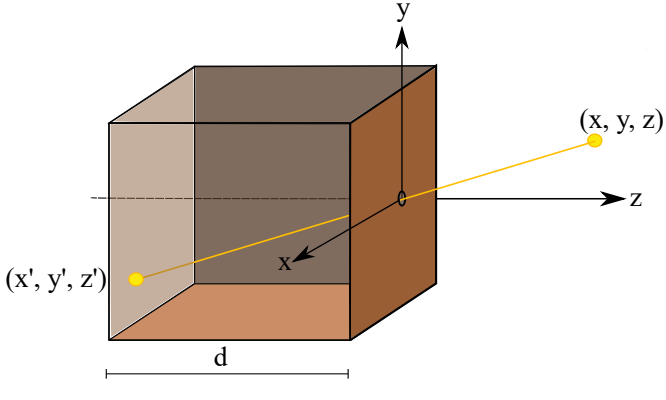


Figura 1.2. Cámara estenopeica. Proyección del punto exterior sobre la película.

Entonces, de todos los rayos que se reflejan desde algún punto $P = (x, y, z)$ en el exterior, solo un rayo, o más precisamente, un haz de luz muy estrecho logra entrar por la apertura, hasta que finalmente choca sobre el material fotosensible en el punto $P' = (x', y', z')$, véase **Figura 1.2**.

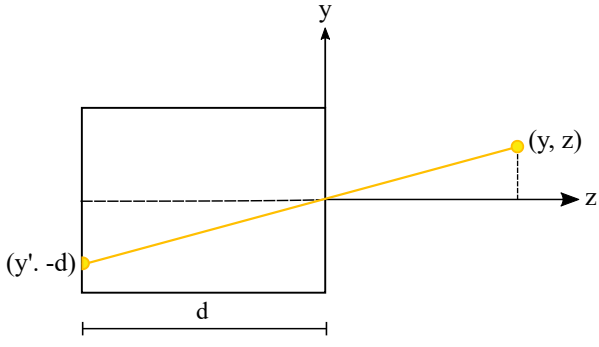


Figura 1.3. Cámara estenopeica (vista de perfil).

Ahora bien, considerando lo que pasa cuando un rayo de luz atraviesa la apertura, de la **Figura 1.3**, podemos observar que los dos triángulos rectángulos que se forman, uno con el punto exterior $P = (x, y, z)$ y otro el punto proyectado P', son semejantes, permitiéndonos calcular la posición del punto proyectado P'. Sabemos que la película se encuentra a distancia $-d$ del origen en z, entonces:

$$y' = -\frac{y}{z/d} \quad \text{y} \quad x' = -\frac{x}{z/d}$$

$$\Rightarrow P' = (x', y', -d)$$

Así el color del punto P' en nuestra película sería el mismo que el del punto exterior P, proyectando la imagen de manera invertida.

Otro aspecto a considerar, es el *ángulo de visión* o *campo de visión*, que es qué tanto de la escena se alcanza a mostrar, o bien, qué tan grandes se ven los objetos del exterior.

Dicho aspecto depende también de la distancia d que hay entre la película y la apertura, conocido como *distancia focal*. Ya que al acercar la película a la apertura, los objetos se hacen más pequeños y una mayor parte de la escena es proyectada, es decir, pareciera que nos alejamos, haciendo más amplio el ángulo de visión.

Y por el contrario, cuando alejamos la película de la apertura (mayor distancia focal), se muestra una porción más pequeña de la escena, es decir, como si nos acercáramos, haciendo más estrecho el ángulo de visión, como podemos notar en la **Figura 1.4**

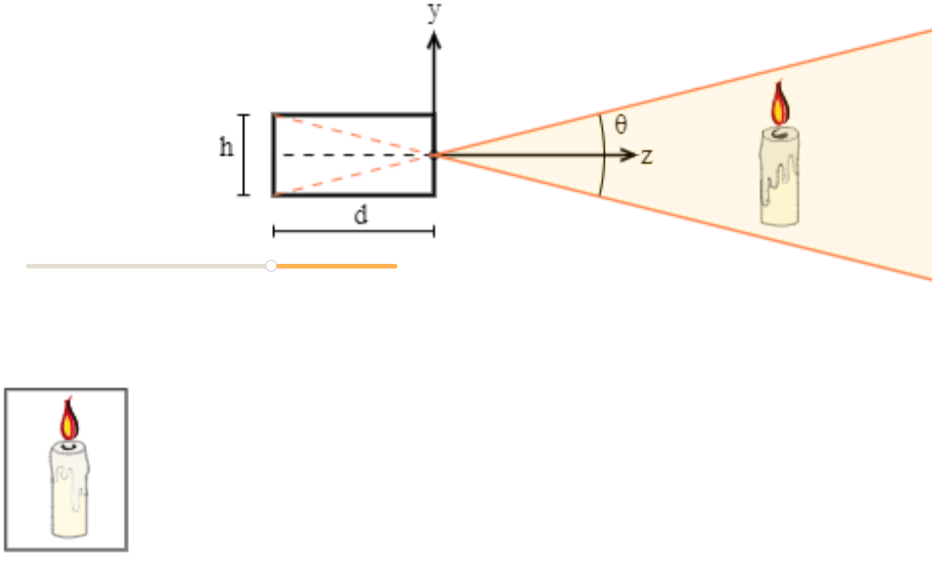


Figura 1.4. Cámara estenopeica. Mueve el slider para modificar la distancia focal d de la cámara y observe su relación con la amplitud el ángulo de visión θ.

Por lo que sí la altura de la cámara es $\boldsymbol{h}$, entonces el ángulo de visión $\boldsymbol{\theta}$ estaría dado por

$$tan\left(\frac{\theta}{2}\right) = \frac{h/2}{d} \quad \Rightarrow \quad \theta = 2tan^{-1}\frac{h}{2d}.$$

Dicho ángulo de visión puede ser definido de manera vertical, conectando la apertura con las esquinas superior e inferior de la película, o bien, de manera horizontal, conectado los extremos derecho e izquierdo de la película.

La cámara pinhole tiene un *campo de profundidad* infinito, esto quiere decir que cada punto en la escena dentro del campo de visión es proyectado dentro de la cámara.

En GC se suele posicionar el plano de proyección o película en $z = d$, que es algo que no funcionaría con una cámara real, pero en el mundo virtual de la computadora sí, evitando así los signos negativos y colocando la imagen en la posición original (de arriba hacia abajo).

Entonces, el punto proyectado estaría dado por

$$P' = (x', y', z') = (\frac{dx}{z}, \frac{dy}{z}, d)$$

Renombrando también al punto estenopeico como el "ojo" o posición de la cámara, o solamente como *viewpoint*; y el plano de proyección o la película, como *image plane*, digamos nuestra imagen final compuesta de pixeles.

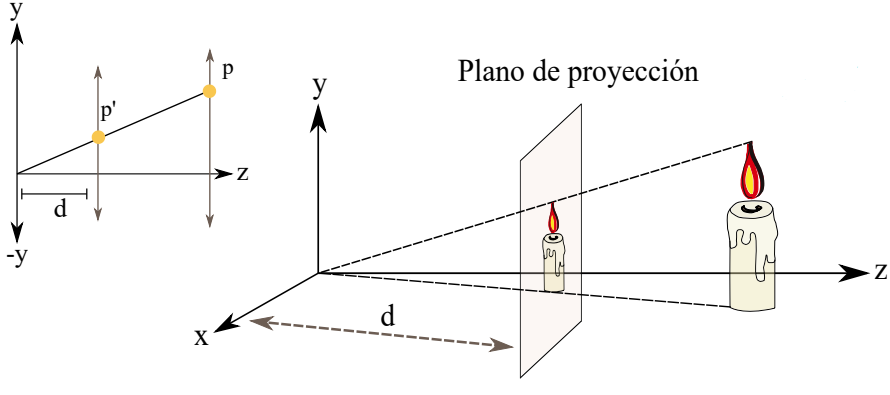


Figura 1.5. Plano de proyección (paralelo eje XY)

Si bien, lo ideal sería que la apertura de la cámara pinhole pudiera capturar un solo rayo de luz por punto, ya que de este modo, habría una única correspondencia entre los puntos en el exterior y los puntos proyectados, obteniendo una imagen nítida como resultado.

Pues de lo contrario, cada punto en el exterior podría proyectarse varias veces y producir una imagen desenfocada o borrosa. Este efecto se puede apreciar más cuando se fotografían objetos pequeños y brillantes con un fondo obscuro, generando círculos borrosos.

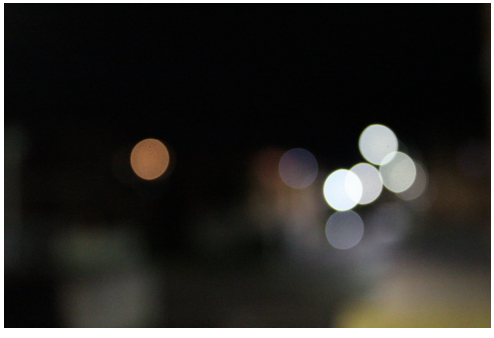


Figura 1.6. Fotografía desenfocada: Alumbrado público de noche. Foto por Raziel Almanza.

No obstante, en el mundo real, no es posible hacer tal apertura, ya que si el agujero es demasiado pequeño (comparable con el tamaño de la longitud de onda de la luz), lo rayos son difractados, sin embargo, se puede encontrar un

diámetro lo suficientemente pequeño para lograr obtener una buena imagen. Por ejemplo, una apertura de 2mm en una caja de zapatos, produce muy buenos resultados.

La cámara estenopeica tiene dos desventajas, una es que al tener una apertura muy pequeña, se requiere de más tiempo de exposición, pues se necesita de cierta cantidad de luz para formar la imagen sobre el material fotosensible; y en consecuencia se produce un efecto borroso si la cámara o algo en la escena exterior se mueve. La segunda es que no es posible modificar la distancia focal de la cámara, para obtener un mayor o menor campo de visión. Cabe mencionar, que éstos no son un problema en el mundo virtual.

Dichas desventajas se pueden solucionar agregando un lente sobre la apertura. Los lentes permiten una mayor apertura, para pasar más luz a la cámara y a su vez, redireccionando los rayos de luz reflejados por un mismo punto en el exterior a puntos únicos sobre la película. Proporcionando una imagen nítida en menor tiempo de exposición, lo cual resuelve el primer punto. Mientras que para el segundo, basta con elegir o ajustar los lentes a una distancia focal deseada.

1.3 El Sistema de Visión Humano

La visión, es quizás el sentido más importante que tenemos, ya que nos proporciona información útil, como percibir increíbles imágenes, así como el movimiento y diferenciar objetos, siendo fundamental para la supervivencia del ser humano a lo largo del tiempo.

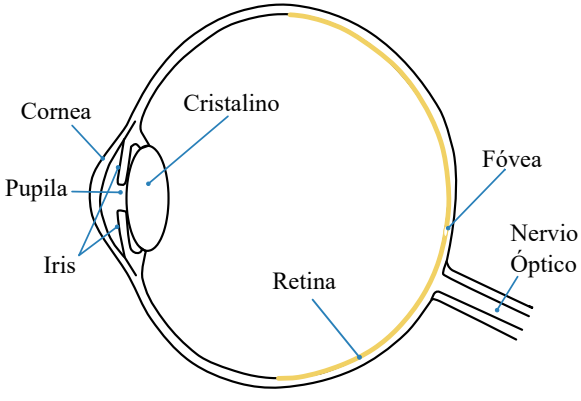


Figura 1.7. Anatomía del ojo humano.

El sistema de visión humano (SVH) nos ayuda a comprender este complejo proceso biológico y psicológico, el cual, a pesar de ser tan complejo, sigue respetando los principios físicos de cualquier otro sistema óptico. Dicho sistema está compuesto por los ojos y el cerebro.

El proceso comienza cuando la luz llega al ojo y entra a través de la cornea (Ver **Figura 1.7**), que a su vez pasa por la pupila, la cual (controlada por el iris) puede cambiar el tamaño de su diámetro de 2mm a 8mm, regulando la cantidad de luz que entra, y que es refractada por el cristalino sobre la retina, proyectando la imagen de manera invertida (como en la cámara *pinhole*).

La retina contiene fotorreceptores o receptores de luz, llamados *conos* y *bastones*, estos receptores transforman la luz en pulsos eléctricos (de acuerdo a su sensibilidad y la intensidad de la luz), los cuales transportados por el nervio óptico, llegarán al cerebro para ser interpretados.

La luz es una forma de radiación electromagnética que se propaga en forma de onda, de la cual el ser humano sólo puede percibir aquella que se encuentre dentro del *espectro visible* (**Figura 1.8**). La luz visible suele ser definida como aquella que tiene longitudes de onda que rondan en un rango de 380nm a 780nm, entre la luz ultravioleta e infrarroja.

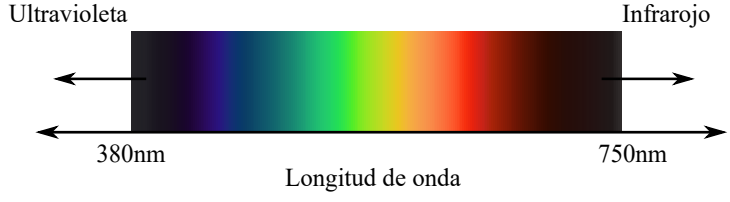


Figura 1.8. Espectro visible por el ojo humano.

Los bastones, son los más numerosos, y son responsables principalmente de nuestra visión nocturna (*visión escotópica*), debido a que pueden captar cantidades muy bajas de luz. Además, son más sensibles a la detección de movimiento o *flicker* y a la visión periférica, debido a su distribución sobre la retina.

Mientras que los conos son menos sensibles, y responden a niveles altos de luz (*visión fotópica*), es decir, durante el día, o bien, con cantidades normales de luz, éstos nos permiten percibir los colores, así como detalles más finos y un cambio de imágenes más rápido, esto último debido a que responden más rápido que los bastones.

Un punto medio, es decir, cuando los niveles de luz se encuentran en un nivel intermedio (*visión mesópica*), ambos receptores son utilizados. Aquí es donde empieza nuestra visión del color, así como la saturación de nuestros bastones. Un ejemplo de la visión mesópica sería, cuando vamos caminando por la calle de noche con alumbrado público.

La mayoría de los conos se concentran en la fóvea, por lo que es la área con mayor agudeza visual. Esta área está libre de bastones, sin embargo, a medida en que nos vamos alejando hacia la periferia del ojo, el número de conos decrece mientras que el número de bastones incrementa. Por esta razón también es que podemos ver una estrella con poca luminosidad al no enfocarnos en ella.

Existen tres tipos de conos: L (*long*), M (*medium*) y S (*short*), que como sus nombres lo indican, están definidos de acuerdo a la parte del espectro visible a la que son más sensibles.

Pues como podemos observar en **Figura 1.9**, la distribución de las respuestas de cada uno de los conos no es uniforme, y abarcan rangos de longitudes de onda que se intersecan entre sí, sin embargo, cada tipo tiene un punto máximo diferente. De modo que, los conos L son más sensibles a las longitudes de onda más largas, correspondiendo a la parte roja del espectro visible, los conos M a las longitudes de onda medianas, siendo la parte verde del espectro y los S a las más cortas, es decir, la parte azul del espectro.

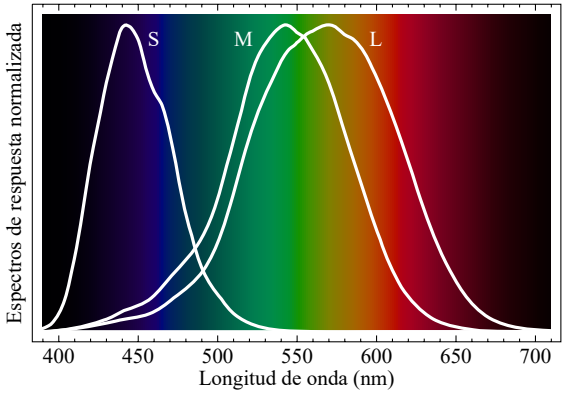


Figura 1.9. Espectros de respuestas normalizada de los conos: S, M y L.

Cabe aclarar que no es que las longitudes de ondas sean de cierto color, puesto que no es una propiedad de la luz, es el cerebro quien las interpreta o produce la sensación del color, así como de tamaño y forma. Por lo que se podría decir que cada cono es capaz de percibir cierto rango de colores.

Por ejemplo, cuando percibimos el color amarillo es porque los conos tipo L se estimulan un poco más que los M. Del mismo modo, cuando percibimos el color azul es porque los conos tipo S se estimulan más. Mientras que para el blanco, todos los conos son estimulados por igual.

Como consecuencia, al tener solo a estos tres tipos de receptores del color (en lugar de tener un tipo de cono distinto para cada una de las longitudes de onda), se simplificó bastante la tarea de manejar los colores en una pantalla, pudiendo del mismo modo aproximarnos a cualquier otro color por medio de los tres primarios.

Otra característica interesante es que los humanos somos más sensibles a los cambios de brillo que de color, el brillo es la intensidad con la que percibimos la luz emitida por un objeto. Por lo que, dicha propiedad ha sido tomada para la compresión de imágenes, donde la información de los colores se comprime más que la información de la iluminación.

1.4 Color



Figura 1.10. La imagen de la izquierda ilustra una escena alegre y contiene con colores con colores cálidos (Birthday vector created by upklyak - www.freepik.com), mientras que la imagen de la derecha contiene colores fríos y muestra una escena triste Kids vector created by brgfx - www.freepik.com.

El color es un aspecto fundamental para la graficación por computadora, ya que además de ser una característica importante en los objetos, nos permite transmitir sensaciones y emociones. Por ejemplo, una escena con colores cálidos puede evocar desde sentimientos de calidez y confort hasta ira u hostilidad, mientras que una escena con colores fríos evoca calma, o bien, sentimientos de tristeza o indiferencia (ver **Figura 1.10**).

Podemos pensar en el color en *dos niveles*. En un nivel físico, que consiste en todas las leyes físicas que se involucran en el SVH para crear la sensación del color, y en un nivel perceptual o subjetivo, que se refiere a cómo lo percibimos.

Y debido a la influencia de los diferentes factores, tanto objetivos como subjetivos, se han creado una variedad de modelos para representar a los colores.

1.4.1 Modelo RGB

El *modelo RGB*, también conocido como *Additive Color Model* o modelo natural, ya que de manera similar a nuestros receptores de colores, se combinan las intensidades de los tres colores primarios (Rojo, Verde y Azul), para obtener un color y es el modelo más utilizado.

Podemos pensarlo como tres lámparas que proyectan los colores primarios sobre una pared blanca, en un cuarto obscuro. De este modo, en la pared blanca se reflejaría el color de las luces, así como la correspondiente combinación aditiva de las luces que estén superpuestas. En especial, al superponer las tres luces, podríamos obtener cualquier color al ir ajustando de manera adecuada la intensidad de cada una.

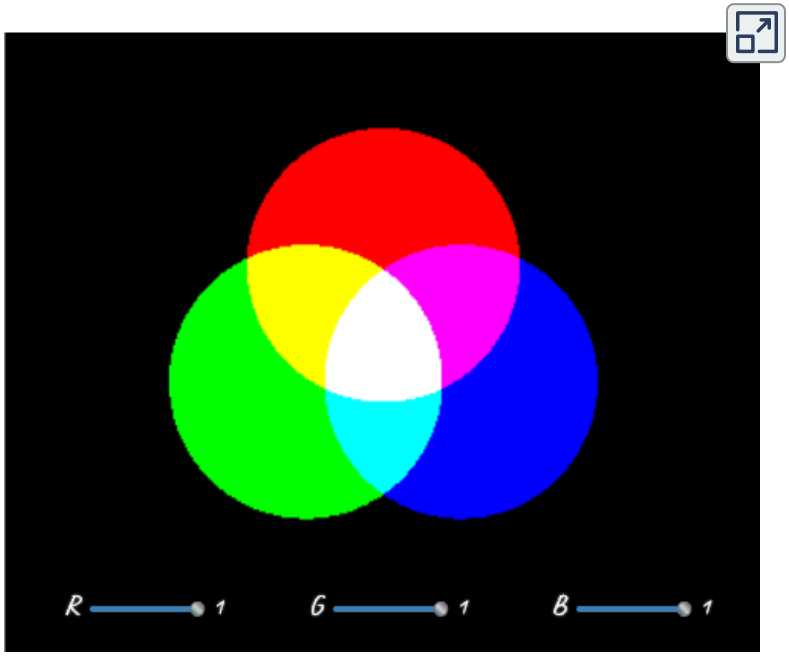


Figura 1.11. Una representación del modelo aditivo utilizando tres luces con los colores primarios sobrepuestas entre sí. Obsérvese que el la luz verde con la luz roja forman el color amarillo, la luz azul con la roja el colo magenta, la luz azul con la verde el color cyan, y las tres se acercan al color blanco. Puedes modificar la intensidad de cada color con los sliders.

Su espacio de color puede ser representado por el cubo unitario $[0,1] \times [0,1] \times [0,1]$ como se muestra en la **Figura 1.12**, donde un color es representado por

una tripleta (R,G,B), haciendo referencia a un punto en el cubo, e indicando al mismo tiempo la intensidad de cada componente. Se asigna el color negro al origen $(0, 0, 0)$, y conforme la intensidad de los colores incremente se llega al color blanco en $(1, 1, 1)$ al moverse a lo largo de los ejes.

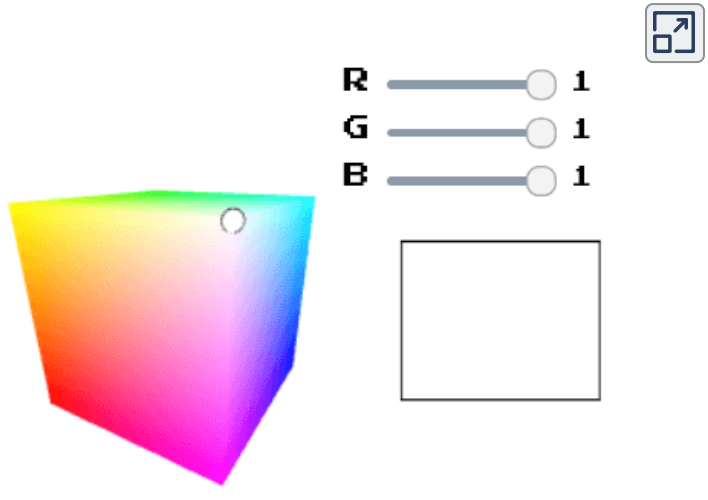


Figura 1.12. Modelo de color RGB. Se invita al lector a mover el modelo con el ratón así como variar los valores R,G y B.

Comúnmente, se utilizan 8-bits por componente, por lo que cada uno puede representar hasta $2^8 = 256$ valores diferentes.

Algunos de los dispositivos en los que se emplea este modelo son las pantallas (*CRT*, *LCD*, *LED*, *OLED*, *AMOLED*), cámaras digitales y escáneres. Por el contrario, la manera en que especifican un color RGB no es muy intuitiva.

1.4.2 Modelo HSV

El *modelo HSV* está basado en el sistema de color de Munsell, el cual intenta acercarse más a la manera en la que percibimos los atributos de los colores, simulando el modo en que un pintor obtiene un color, al mezclar el pigmento con color blanco y/o negro, con el fin de obtener un color más claro, más obscuro u opaco.

Su espacio de color es modelado con un cono o una pirámide hexagonal inversa, y como su nombre lo indica, está definido por sus componentes:

- **H (*Hue*)** es un ángulo entre 0° y 360°, donde cada valor corresponde a un color "puro" (en el espectro visible) o pigmento en la circunferencia del modelo. Empezando con el color rojo en 0° y el verde en 120°.
- **S (*Saturation*)** $\in [0, 1]$, se relaciona con la pureza o intensidad del color, es decir, qué tanto vamos a diluir o rebajar el pigmento con un tono entre los colores blanco y negro, definido por el último componente. Donde 1 es el color puro y 0 es el tono en **V**, correspondiendo a la generatriz del cono, o bien, el radio del cilindro.
- **V (*Value*)** $\in [0, 1]$, donde 0 es el color negro y 1 el blanco, haciendo referencia al factor de brillo, correspondiendo al eje del modelo.

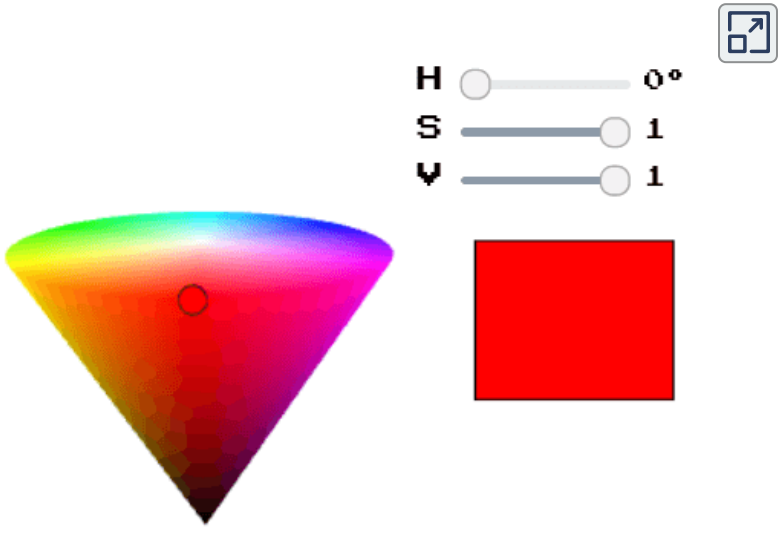


Figura 1.13. Modelo de color HSV. Se invita al lector a mover el modelo con el ratón así como variar los valores H,S y V.

1.4.3 Espacio de color y *gamut*

Cuando un modelo de color es asociado con una función que mapea los colores reales o físicos a los elementos del modelo, se le conoce como *espacio de color*.

Existen espacios de colores que son capaces de representar cualquier color, sin importar la manera en la que serán físicamente reproducidos, estos se conocen como *espacios de color independientes de los dispositivos* . Por otro lado, los *espacios de colores dependientes de los dispositivos*, tienen que lidiar con el hecho de que no todos los colores físicos pueden ser reproducidos.

De modo que, un dispositivo en particular puede producir un conjunto de colores específico, conocido como *gamut* o *gama de colores reproducibles*. Por lo que los colores que no se encuentren dentro de la gama, no podrán ser reproducidos de manera adecuada.

El *gamut* de los espacios de colores dependientes, suelen ser definidos en el diagrama de cromaticidad del espacio de color estándar CIEXYZ (Ver **Figura 1.14**), creado por la *Comission International de l'Eclairage* en 1931. El cual es un espacio de color independiente, basado en un experimento científico, que

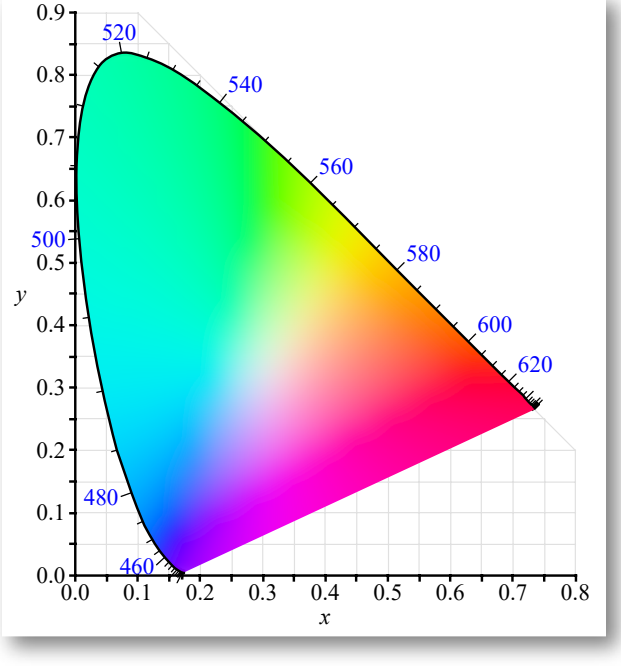


Figura 1.14. Diagrama de cromaticidad del espacio de color The CIE 1931.

engloba todos los colores del espectro visible que en promedio podemos distinguir. Estableciendo los tres colores primarios a partir de los cuales se generan todos los demás.

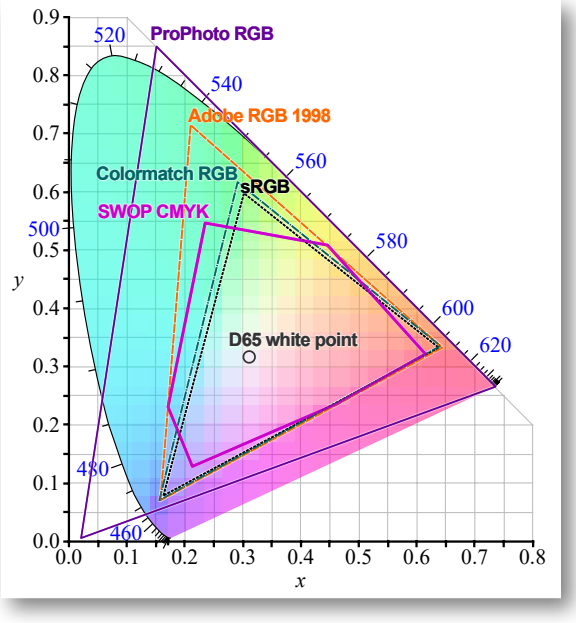


Figura 1.15. Comparación del gamut entre espacios de colores sobre el diagrama de cromaticidad del espacio de color The CIE 1931. Autor: BenRG and cmglee. Licencia.

Por ejemplo los espacios *sRGB* y *AdobeRGB*, son dos espacios de colores distintos asociados a un mismo modelo de color, donde la tripleta RGB(21,03,97) podría mostrar un color diferente en cada espacio. Pues como podemos notar en la **Figura 1.15**, a pesar de que comparten los colores primarios rojo y azul, el verde es distinto, haciendo que la gama de colores del espacio de color *AdobeRGB* sea más amplia que la del *sRGB*.

Otro ejemplo en donde encontramos una diferencia a simple vista, es entre la gama de colores de una impresora de oficina y un monitor, donde el rango de colores de la impresora está mucho más limitada.

1.5 El pipeline gráfico

El *pipeline gráfico* o *rendering pipeline* es el proceso que se requiere para renderizar o generar una imagen 2D, dado un conjunto de datos de entrada que describen un escena tridimensional.

El pipeline se divide en varias etapas, las cuales cumplen con cierta parte del proceso; estas etapas son ejecutadas en paralelo, pero cada una depende del resultado de la etapa previa.

Podemos pensarlo como el proceso dentro de una fábrica ensambladora, o bien, como la analogía que nos dan en [4], de una cocina de comida rápida. En donde para preparar rápido un gran número de sándwiches, se dividen las tareas entre varias personas, de manera que la primera le unta el aderezo al pan, la segunda le pone el jamón, y así consecutivamente; de modo que cada etapa espera a que la etapa previa haya terminado, para realizar su tarea y continuar con el proceso.

Si bien, no hay una sola forma de llevar a cabo el pipeline, mencionaremos las 4 etapas fundamentales: *Aplicación*, *Procesamiento Geométrico*, *Rasterización* y *Procesamiento de pixeles*, dentro de las cuales también tienen sus propios pipelines, es decir, se subdividen en varias etapas. REF.[4].

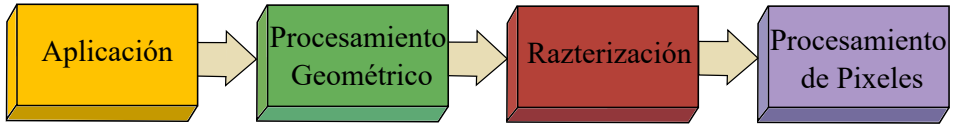


Figura 1.16. Etapas del Pipeline Gráfico

Esta estructura es el motor del pipeline gráfico, y es usada en aplicaciones en tiempo real.

Vale la pena mencionar, que en este tipo de renderizado, también llamado renderizado por rasterización, es un proceso distinto al de *ray tracing*, donde por cada pixel se calcula su respectivo color de acuerdo a los objetos que lo intersecan. Ya que en el renderizado por rasterización se considera a cada objeto y se buscan los pixeles que lo conforman.

A continuación, explicaremos un poco de lo que trata cada etapa del pipeline (sin meternos en detalles de implementación).

1.5.1 Aplicación

La *Aplicación*, como su nombre lo indica esta etapa es ejecutada por la aplicación misma, es decir, el software que es ejecutado desde el CPU. Aquí es el programador quien tiene todo el control, pues es el que se encarga de definir las funcionalidades que ésta va a tener, por lo mismo, es también, donde se pueden hacer optimizaciones en el desempeño de dicha aplicación.

En esta etapa es donde son implementadas otras tareas de acuerdo a los objetivos del desarrollador, como detectar colisiones entre dos objetos, definir animaciones o simulaciones de modelos físicos, entre otras. Así como la implementación de algoritmos de aceleración.

También es la que se encarga de manejar la interacción con otros dispositivos de entrada, definiendo las acciones a ejecutar de acuerdo a la entrada, por ejemplo, cuando alguien presiona una tecla o mueve el cursor del ratón.

Y es donde se especifica la geometría que se quiere renderizar, para ser procesada en a la siguiente etapa. Dicha geometría es especificada con primitivas de renderizado, las cuales pueden ser líneas, puntos o triángulos que finalmente podrían ser mostradas en la pantalla.

A partir de la siguiente etapa las operaciones son realizadas desde el GPU.

1.5.2 Procesamiento Geométrico (o de vértices)

El *Procesamiento geométrico o de vértices*, se encarga de convertir los vértices en primitivas, realizando la mayoría de las operaciones por vértice y por triángulo, dicho proceso se describe en el siguiente diagrama.

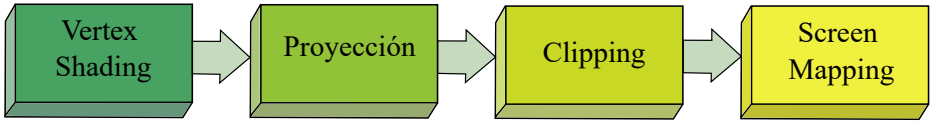


Figura 1.17. Etapas en el procesamiento geométrico.

En el *Vertex shading* se efectuan la mayoría de las transformaciones para calcular las posiciones de los vértices y de otros datos o atributos asociados a éstos, como serían las normales de los vértices o algún otro vector que se quiera calcular, así como las coordenadas de textura o un color.

Para empezar, tenemos de entrada la descripción geometrica de cada uno de los modelos que queremos, es decir, por cada objeto, tenemos un conjunto de vértices ordenados, los cuales forman o describen a dichos objetos, y se encuentran descritos en su propio espacio local (**Figura 1.18**).

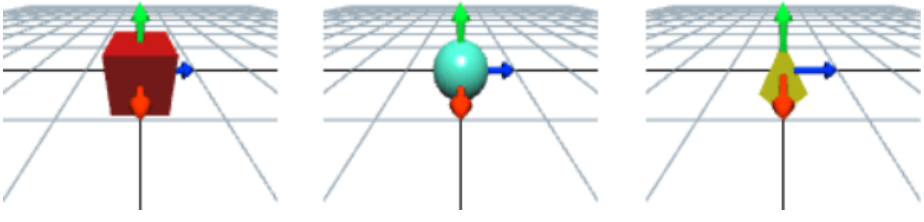


Figura 1.18. Espacio local. Por lo regular los objetos son descritos con su punto céntrico en el origen. Puedes mover la posición de la cámara con el ratón o dedo para tener una mejor vista.

Las coordenadas de un objeto se conocen como coordenadas locales, sobre las cuales se pueden aplicar una serie de transformaciones para modelar al objeto (escalarlo, rotalo y/o trasladarlo), a estas transformaciones se les llaman transformaciones de mundo o de modelo (**Figura 1.19**).

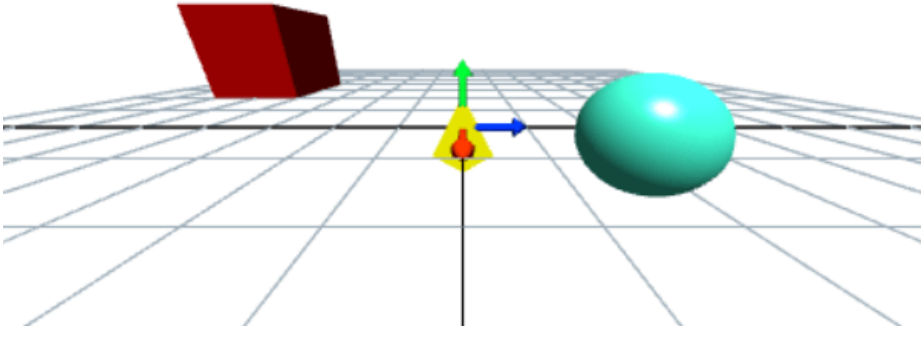


Figura 1.19. Transformación de modelo. Se transforman las coordenadas locales del objeto a coordenadas globales.

Una vez que éstas han sido aplicadas, se dice que los objetos se encuentran posicionados en el espacio global o de mundo, donde los objetos ya están ordenados dentro de un mismo espacio.

La siguiente transformación a aplicar es la *transformación de vista*, ésta es la que modela la cámara, ya que lo que nos interesa renderizar será lo que la cámara (o el observador) esté viendo. Efectuando un cambio de base de las coordenadas del mundo a las coordenadas de la cámara, es decir, del espacio global al espacio de la cámara, también llamado *view space* o *eye space*. Colocando a la cámara en el origen, mirando comunmente en dirección al eje z negativo ($z-$), con el eje y positivo ($y+$) apuntando hacia arriba , y el eje x positivo ($x+$) a la derecha (**Figura 1.20**).

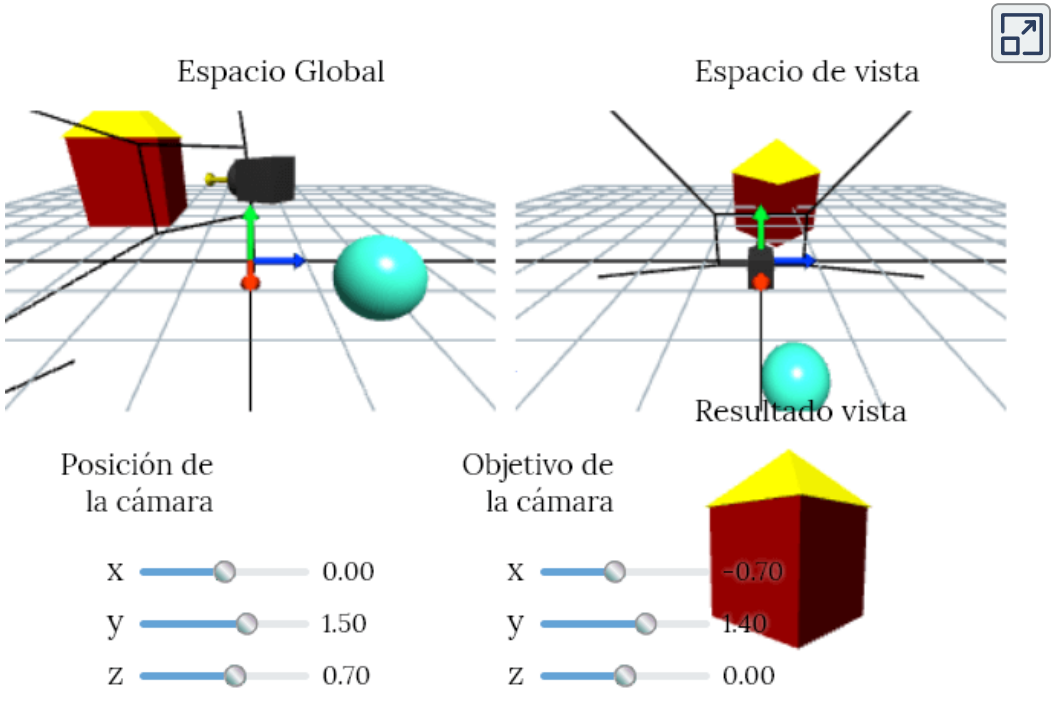


Figura 1.20. Transformación de vista. En la visualización superior izquierda se muestra la cámara dirigida y posicionada en donde queremos que esté dentro del espacio global. En la visualización superior derecha se muestra cómo todos los objetos son reorientados al espacio de la cámara de modo que la cámara termine en el origen y apuntando hacia $z+$. En la última visualización se muestra lo que la cámara esta viendo de acuerdo con la posición y dirección definida. Se considera la proyección de perspectiva para visualizar los objetos, y su volumen de vista se describe por el frustrum de las escenas. Se invita al lector a modificar los valores de la posición y dirección de la cámara. Amplia el interactivo para poder observar mejor lo que ocurre. Puedes mover la posición de la cámara de las escenas superiores con el ratón o dedo, así como alejarte o acercarte con la rueda del ratón para tener una mejor noción del espacio.

Hasta aquí ambas transformaciones (de modelo y de vista) también suelen ser aplicadas a las normales de los vértices, así como otros véctores que ayuden a modelar el aspecto del objeto. Por esta razón, la parte programable del procesamiento de vértices es llamada como *vertex shader*. Y dichas transformaciones pueden ser implementadas con matrices de 4x4, las cuales veremos también en el Capítulo 4 y 5.

Sin embargo, tanto la posición de los vértices como de los otros atributos pueden ser calculadas o modelados como el programador prefiera.

La siguiente etapa es la de *Proyección*, y en ésta se efectúa otra transformación, la cual también puede ser efectuada con una matriz 4×4, por lo que suele ser concatenada a las matrices de transformación anteriores dentro del vertex shader. Este tipo de matrices de transformación son llamadas de proyección ya que lo que se busca es proyectar lo que se está viendo en la pantalla de la cámara, de manera similar a lo que ocurre con la cámara pinhole.

En esta etapa se define el volumen de visión que ve nuestra cámara, el cual define el espacio de la escena a renderizar, y que será transformado en un cubo unitario, también llamado *volumen canónico de vista*, el cual suele tener sus extremos en $(-1, -1, -1)$ y $(1, 1, 1)$. Para ello, se elige un tipo de proyección con la que se definirán los límites del volumen de visión o *clipping volume* y se transformará en el cubo unitario, para la conveniencia de las etapas posteriores, pasando las coordenadas de vista a coordenadas de *clipping* o de recorte. Los dos tipos de proyección más utilizados son la proyección ortográfica y la de perspectiva, obsérvese la **Figura 1.21**.



Figura 1.21. Proyección Ortgonal (izquierda) y Proyección de Perspectiva (derecha).

En la primera se utiliza un volumen de visión con forma de prisma rectangular, en esta los puntos son proyectados de manera paralela y el volumen de visión ortogonal es transformado en un cubo unitario. Por otra parte, la proyección de perspectiva tiene un volumen de visión con forma de pirámide truncada o *frustrum* como el que vimos en la **Figura 1.20**, este tipo de proyección nos proporciona una sensación más realista de profundidad, ya que mientras más alejado se encuentre un objeto de la cámara más pequeño se verá. Y del mismo modo, el frustrum es transformado a un cubo unitario.

Con ésta última transformación se finaliza el proceso del vertex shader.

El objetivo del *Clipping* es descartar total o parcialmente aquellas primitivas o partes de primitivas que queden fuera del volumen de visión. Pues recordemos que lo que nos interesa renderizar es lo que esté adentro de éste, por lo que sí la primitiva se encuentra completamente dentro del volumen entonces pasa a la siguiente etapa para ser renderizada y por el contrario, si se encuentra completamente fuera es descartada. Por otro lado, sí la primitiva se encuentra parcialmente dentro, entonces es recortada, generando nuevos vértices y primitivas a partir de la intersección con el volumen y los vértices que quedan fuera de éste son descartados (**Figura 1.22**).

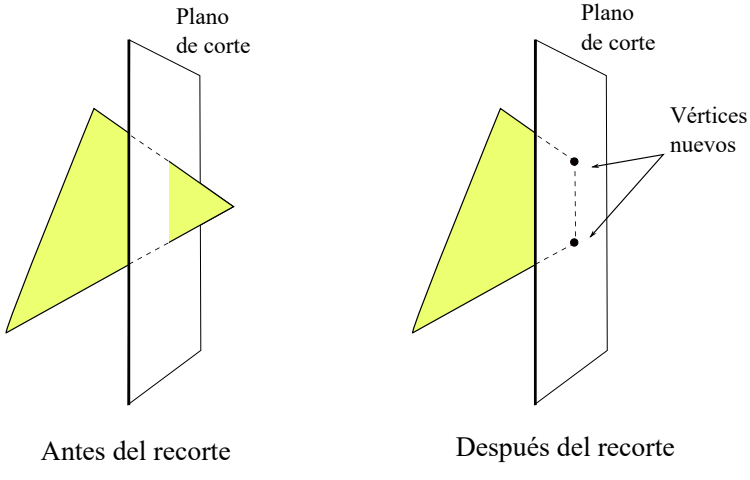


Figura 1.22. Clipping. Un polígono es cortado con respecto a un plano del volumen de visión, la porción que queda afuera es recortada del poligono, generando así nuevos vértices, mientras que la que queda dentro es retenida para la siguiente etapa.

Finalmente, se normalizan las coordenadas de clipping con la división de perspectiva, al dividir las coordenadas entre su última componente w, convir-

tiéndolas a *coordenadas normalizadas del dispositivo o NDC* (*Normalized Device Coordinates*). El término de normalizadas en NDC viene de que las coordenadas de los puntos están dentro del rango de $[-1, 1]$.

Como última etapa del procesamiento geométrico tenemos al *Screen mapping*, también conocida como transformación de pantalla o *viewport*, que como su nombre lo indica, se encarga de transformar las coordenadas x y y en coordenadas de pantalla o *window coordinates*, las cuales son pasadas a la etapa de rasterización, en conjunto con su coordenda z.
Dicha transformación es efectuada de acuerdo a las coordenadas de pantalla de la ventana sobre la que se renderizará la escena.

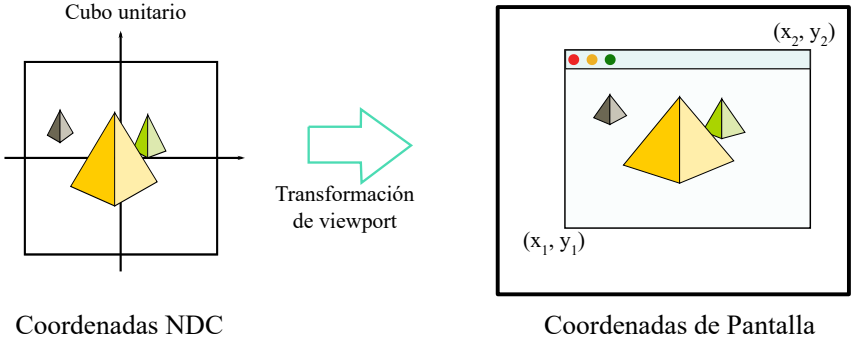


Figura 1.23. Screen Mapping. Se transforman las coordenadas de clipping a coordenadas de pantalla: Teniendo como extremo inferior de la ventana en (x_1, y_1) y el extremo contrario superior en (x_2, y_2), con $x_1 < x_2$ y $y_1 < y_2$.

1.5.3 Rasterización

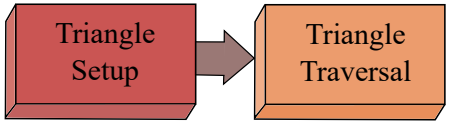


Figura 1.24. Etapas en la rasterización.

Esta etapa es también conocida como *scan conversion*, y es la responsable de convertir los vértices de coordenadas de pantalla, y otra información asociada a éstos, en pixeles. Este proceso consta de dos etapas.

La primera etapa, es el *Ensamblado de primitivas* o *Triangle setup*, que se encarga simplemente de ensamblar conjuntos de vértices en primitivas.

Y en la segunda, *Triangle traversal*, es donde se buscan aquellos pixeles que se encuentran dentro de una primitiva, o bien, que por lo menos su centro lo esté; generando un fragmento por cada uno de éstos, los cuales podemos pensarlos como un posible pixel con una información asociada, la cual es resultado de la interpolación entre los vértices a lo largo de la primitiva. Dicha información guarda la profundida de pixel (su valor en z) y cualquier otro dato del vertex shader.

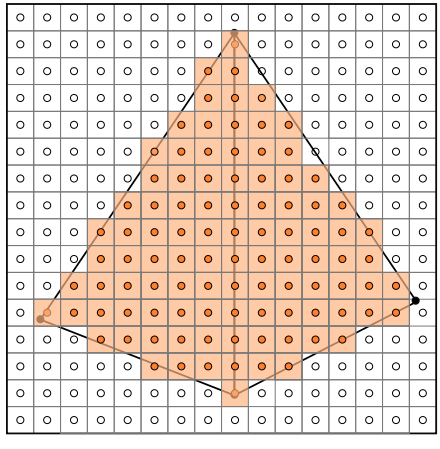


Figura 1.25. Triangle Traversal. Se determinan los pixeles de la pantalla que cubren una primitiva.

Finalmente dichos fragmentos (o pixeles) son enviados a la siguiente y última etapa.

1.5.4 Procesamiento de pixeles (o de fragmentos)

Es la última etapa del pipeline, y también se puede dividir en dos etapas: *Pixel shading* y *Merging*.

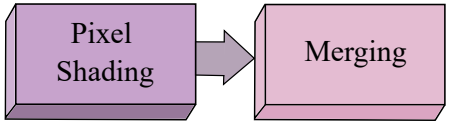


Figura 1.26. Etapas en el procesamiento de fragmentos.

En el *Pixel shading* es donde se realizan las operaciones del fragment shader, utilizando la información asociada de cada fragmento; y pasando finalmente el color (o los colores) resultantes de cada pixel.
Existen varias técnicas para producir el color de los pixeles; la más importante es el mapeado de texturas, la cual consiste en aplicar una textura sobre un objeto, en otras palabras, es como si pegáramos o envolviésemos con una o varias imágenes sobre un objeto.

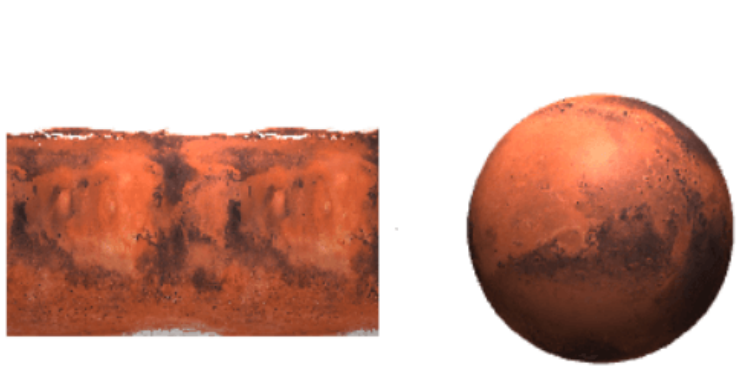


Figura 1.27. Ejemplo de mapeo de textura sobre una esfera. Texturas del planeta Marte obtenidas de https://www.solarsystemscope.com/ bajo la Licencia CC BY 4.0.

Y por último tenemos el *Merging*, y en éste, prácticamente se determina el color final que tendrá cada pixel en la pantalla. Dichos colores son almacenados en el búfer de color, el cual es un arreglo bidimensional, y es inicializado con un color inicial o color de fondo.

Durante esta etapa se resuelve la visibilidad de los fragmentos de acuerdo a su valor de profundidad, es decir, los colores almacenados en el búfer de color deberá contener los colores de las primitivas que son visibles. Comúnmente ésto es calculado con el algoritmo del *z-buffer*.

También se pueden realizar otros efectos para determina el color final de cada pixel, como el de transparencia, donde se aplica una función de mezclado o *blending* que indica cómo se combinan los colores con transparencia. O bien, una operación de plantilla o *stencil* en el que se descartan ciertos fragmentos para después ser pasados por el test de profundidad (y descartar más).

Finalmente el contenido del búfer de color es desplegado en la pantalla.

1.6 Referencias bibliográficas

[1] Shirley, Peter, et al. ***Fundamentals of Computer Graphics.*** A K Press. 3ª Edición. 2009.

[2] Ganovelli, Fabio, et al. ***Introduction to Computer Graphics a Practical Learning Approach.*** CRC Press. 2015.

[3] Angel, Edward, et al. ***Interactive Computer Graphics a Top-Down Approach with WebGL.*** Pearson Education, Inc. 7ª Edición. 2014.

[4] Akenine-Möller, Tomas, et al. ***Real-Time Rendering.*** A K Peters/CRC Press. 4ª Edición. 2018.

[5] Eck, David. ***Introduction to Computer Graphics.*** Version 1.2. 2018. http://math.hws.edu/graphicsbook

[6] Max K. Agoston. ***Computer graphics and geometric modeling implementation and algorithms.*** Springer. 1ª Edición. 2005.

[7] Ammeraal, Leen, et al. ***Computer Graphics for Java Programmers.*** Springer-Verlag. 3ª Edición. 2017.

[8] ***Computational photography***. 23 de Enero del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Computer_simulation&oldid=934831377

[9] ***3D Viewing: the Pinhole Camera Model***. Scratchapixel. Recuperado el 26 de Febrero del 2020 de https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera

[10] ***Computer simulation***. 8 de Enero del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Computer_simulation&oldid=934831377

[11] ***Virtual Reality***. 26 de Febrero del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Virtual_reality&oldid=942740811

[12] ***Realidad aumentada***. 16 de Marzo del 2020. Wikipedia. https://es.wikipedia.org/w/index.php?title=Realidad_aumentada&oldid=124304756

[13] ***Visual system***. 5 de Abril del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Visual_system&oldid=946202371

[14] ***Human visual system model***. 13 de Febrero del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Human_visual_system_model&oldid=940628106

[15] ***Visión***. 2 de Abril del 2020. Wikipedia. https://es.wikipedia.org/w/index.php?title=Visi%C3%B3n&oldid=124802496

[16] Gonzáles, Vicente. 27 de Septiembre del 2014. ***El Sistema Visual Humano***. UAL. https://w3.ual.es/~vruiz/Docencia/Apuntes/Perception/Sistema_Visual/index.html

[17] ***Trichromacy***. 18 de Febrero del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Trichromacy&oldid=941417876

[18] ***Visión mesópica***. 30 de Agosto del 2019. Wikipedia. https://es.wikipedia.org/w/index.php?title=Visi%C3%B3n_mes%C3%B3pica&oldid=118741381

[19] ***Cone cell***. 20 de Mayo del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Cone_cell&oldid=957858342

[20] Nave, Carl R. 2001. ***Rods and Cones***. Hyperphysics. http://hyperphysics.phy-astr.gsu.edu/hbase/vision/rodcone.html

[21] CJ Kazilek, Kim Cooper. 6 de Junio del 2006. ***Rods and Cones of the Human Eye***. ASU - Ask A Biologist. https://askabiologist.asu.edu/rods-and-cones

[22] Kusyk, Janusz., Czudec, Pawel. s. f. ***Color Perception***. StonyBrook - CS. https://www3.cs.stonybrook.edu/~lori/classes/ColorPerception/addproperties.html

[23] Peter K. Kaiser. 1996. ***The Joy of Visual Perception***. YORKU. http://www.yorku.ca/eye/toc.htm

[24] ***Rod cell***. 12 de Abril del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Rod_cell&oldid=950548955

[25] ***Light***. 16 de Abril del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Light&oldid=951305118

[26] ***Color model***. 4 de Febrero del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Color_model&oldid=939112221

[27] ***RGB color model***. 27 de Mayo del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=RGB_color_model&oldid=959272315

[28] ***Colorizer***. http://colorizer.org/

[29] Henderson, AJ. Klecker, Hermann. 26 de Marzo del 2014. ***Re: Difference or relation between a Color Model and a Color Space?***. [Comentario en el foro *Difference or relation between a Color Model and a Color Space?*] Stackexchange. https://photo.stackexchange.com/questions/48984/what-is-the-difference-or-relation-between-a-color-model-and-a-color-space

[30] ***CIE 1931 color space***. 26 de Mayo del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=CIE_1931_color_space&oldid=959024702

[31] Madsen, Rune. s. f. ***Color models and color spaces***. Programming Design Systems. https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html

[32] Berkenfeld, Diane. Black, Dave. s.f. ***Understanding Focal Length***. NikonUSA. Recuperado 10 de Agosto del 2020 de https://www.nikonusa.com/en/learn-and-explore/a/tips-and-techniques/understanding-focal-length.html#

[33] Ward, Caleb. 5 de Junio del 2014. ***Understanding Lenses: What is Focal Length?***. The Beat. https://www.premiumbeat.com/blog/understanding-lenses-what-is-focal-length/

[34] Alamia, Marco. s.f. ***Article - World, View and Projection Transformation Matrices***. Codinglabs. Recuperado 15 de Agosto del 2020 de http://www.codinglabs.net/article_world_view_projection_matrix.aspx

[35] ***Rendering Pipeline Overview***. 8 de Abril del 2019. Kronos. https://www.khronos.org/opengl/wiki_opengl/index.php?title=Rendering_Pipeline_Overview&oldid=14511

[36] ***RGB and HSV/HSI/HSL Color Space Conversion***. s.f. Vocal. https://www.vocal.com/video/rgb-and-hsvhsihsl-color-space-conversion/

[37] Cherry, Kendra. 28 de Mayo del 2020. ***Color Psychology: Does It Affect How You Feel?***. Verywellmind. https://www.verywellmind.com/color-psychology-2795824#:~:text=These%20warm%20colors%20evoke%20emotions,feelings%20of%20sadness%20or%20indifference

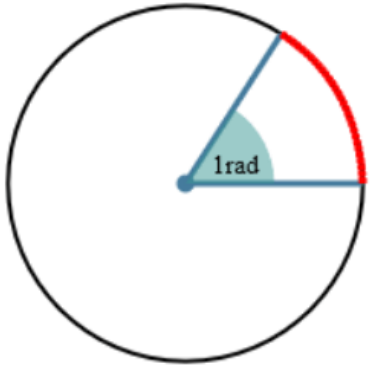
CAPÍTULO II

Conceptos preliminares de álgebra lineal

Recordaremos algunos conceptos de álgebra lineal necesarios para tener una mejor comprensión en los temas posteriores.

2.1 Ángulos, grados y radianes

Los ángulos pueden ser medidos ya sea en grados o radianes. La medida de radianes de un ángulo θ en un círculo con radio r se define como el número de "radios" que caben dentro del arco s formado por θ. Esto es, $\theta = \frac{s}{r} \Rightarrow s = \theta r$.



De modo que sí $r = 1$, entonces el ángulo θ es justamente la longitud del arco definida por el mismo ángulo al cortar el círculo unitario. Por otro lado, usando grados, tenemos que una vuelta completa equivale a 360°, y que al subdividirla podemos obtener el resto de los ángulos.

Entonces, $360^\circ = 2\pi$ radianes, siendo

$$180^\circ = \pi \text{ radianes}$$

Por lo que,

$$1\ radián = \frac{180}{\pi} \approx 57.3 \text{ grados} \quad \text{y} \quad 1\ grado = \frac{\pi}{180} \approx 0.017 \text{ radianes.}$$

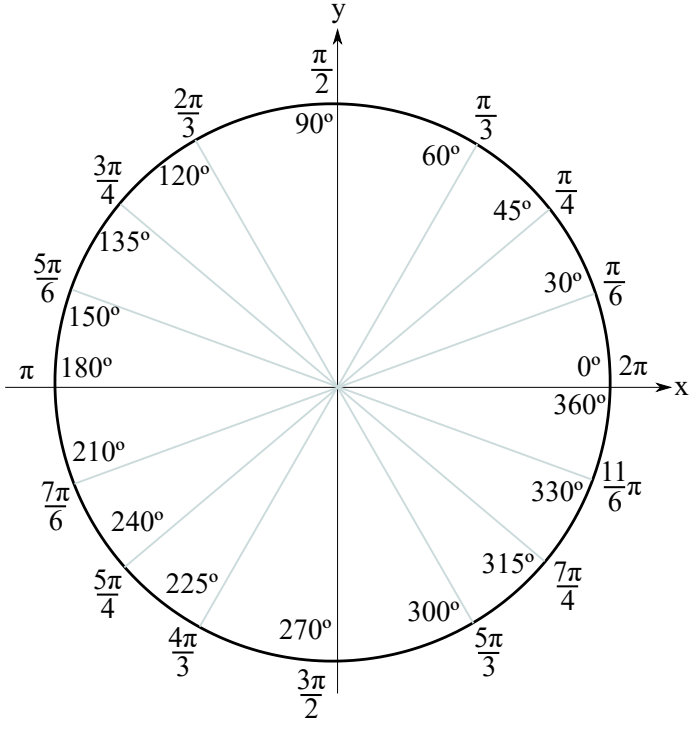


Figura 2.1. Relación entre grados y radianes en un círculo unitario.

Los radianes son la medida más utilizada para realizar cálculos, en particular para llevar acabo funciones trigonométricas. Por último, recordemos que los ángulos positivos se miden en sentido contrario a las manecillas del reloj, a partir del eje $x+$; y para los ángulos negativos es con respecto al sentido de las manecillas.

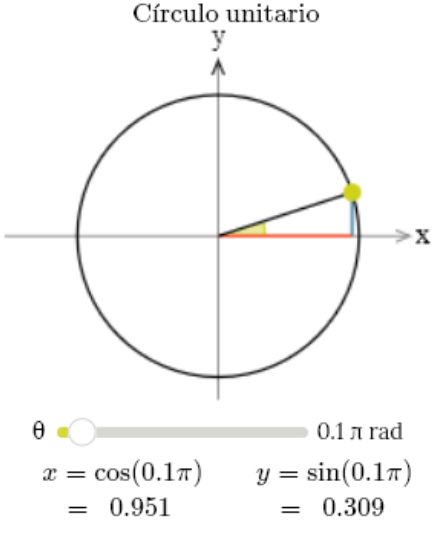
2.2 Un poco de trigonometría

Veamos las funciones trigonométricas básicas. Para cubrir cualquier tipo de ángulo, trazaremos el ángulo dentro de un círculo de radio r, y estará definido por un punto $P = (x, y)$.

$$\textbf{Seno}: \sin\theta = \frac{y}{r} \quad ; \quad \textbf{Coseno}: \cos\theta = \frac{x}{r}$$
$$\textbf{Tangente}: \tan\theta = \frac{y}{x} \quad ; \quad \textbf{Cotangente}: \cot\theta = \frac{x}{y}$$
$$\textbf{Cosecante}: \csc\theta = \frac{r}{y} \quad ; \quad \textbf{Secante}: \sec\theta = \frac{r}{x}$$

Cuando $r = 1$ podemos simplificar las igualdades como sigue:

$$\sin\theta = \frac{y}{1} = y \quad ; \quad \cos\theta = \frac{x}{1} = x$$
$$\tan\theta = \frac{\sin\theta}{\cos\theta} \quad ; \quad \cot\theta = \frac{1}{\tan\theta}$$
$$\csc\theta = \frac{1}{\sin\theta} \quad ; \quad \sec\theta = \frac{1}{\cos\theta}$$



Las siguientes identidades trigonométricas con dos ángulos arbitrarios α y β también son usadas:

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$

$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$

2.3 Vectores

Los vectores son fundamentales en GC, en especial para los gráficos en 3D. Ya que pueden ser utilizados para representar puntos, direcciones, velocidades, desplazamientos, etc. Aprovechando en su totalidad la definición tanto geométrica como algebráica de un vector.

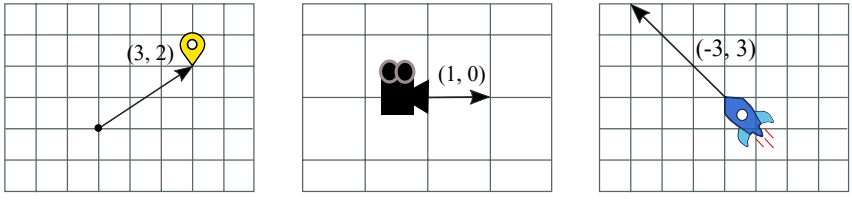


Figura 2.2. a) Posición: índica los pasos o el desplazamiento necesario para alcanzar un objeto desde un punto específico en el mapa, siendo 3 pasos a la derecha y 2 hacia el norte. b) Dirección: una cámara mirándo hacia la derecha. c) Velocidad: una nave espacial que se desplaza -3km a la izquierda y 3km a hacia arriba en un segundo.

Comúnmente para contabilizar objetos basta utilizar un solo número, por ejemplo, para medir la temperatura, la altura, el calzado, etc. Éste tipo de medidas son descritas únicamente por su magnitud y son representadas por número reales, llamados *escalares*, y claro su unidad base. Por otra parte existen fenómenos físicos que necesitan más que un solo número para ser medidos, como serían el movimiento del viento, la fuerza, la velocidad, entre otros. Los cuales poseen magnitud y dirección, éstos son llamados *vectores*.

En este libro los puntos se denotarán con letras mayúsculas en itálicas.

Podemos definir entonces a un *vector* como un segmento de recta que tiene una cierta magnitud y dirección. El cual es representado como una flecha que va de un punto inicial P a un punto terminal Q, y se denota como:

$$\mathbf{v} = \overrightarrow{PQ}$$

Si dos segmentos de recta dirigidos tienen la misma magnitud y dirección, entonces se dice que son *equivalentes*, esto es, sin importar la posición inicial de cada uno.

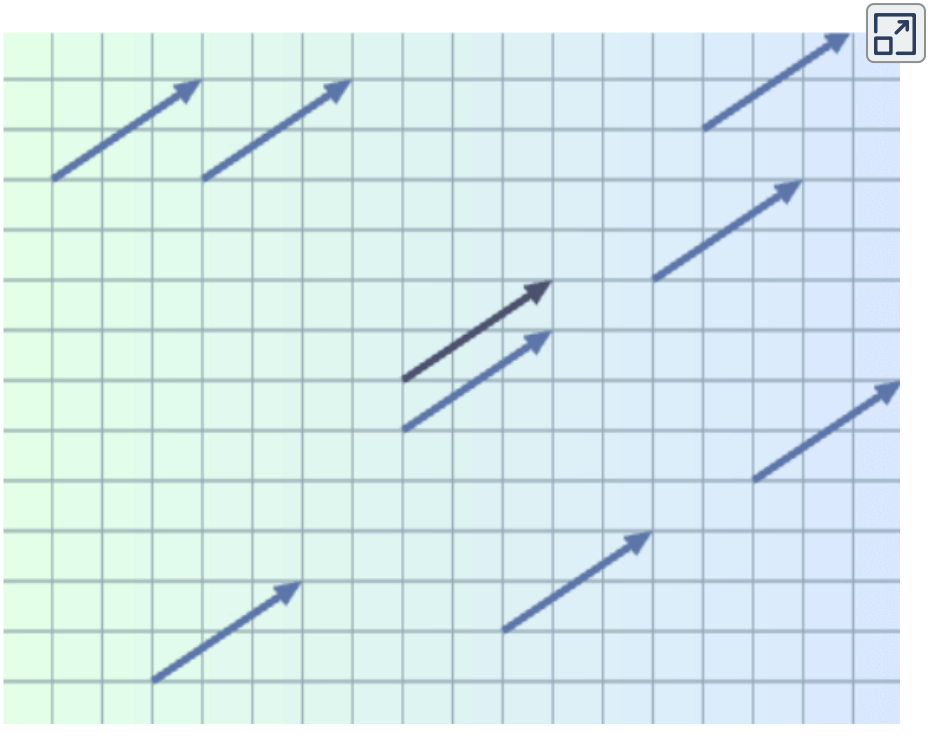


Figura 2.3. Vectores equivalentes: Todas las flechas representan a un mismo vector, puesto que un vector no tiene una posición específica.

Siendo más específicos, podemos ver a un vector como el conjunto de todos los segmentos dirigidos que son equivalentes entre sí.

La *magnitud* o *longitud de un vector* se denota como $\|\mathbf{v}\|$ y es un escalar; más adelante veremos cómo es que se calcula.

Observemos que los vectores $\vec{PQ}$ y $\vec{QP}$ son distintos, puesto que apuntan en direcciones opuestas, sin embargo, éstos siguen teniendo la misma longitud. Por otro lado, si tenemos $\vec{PP}$, entonces el vector tiene longitud cero, es decir, $\|\vec{PP}\| = 0$, ya que su punto inicial y terminal son el mismo, se dice que tampoco tiene dirección. A este vector, se lo conoce como el *vector cero*, y es denotado por $\mathbf{0}$.

2.3.1 Multiplicación de vectores por un escalar

Cuando multiplicamos a un vector $\mathbf{v}$ por un escalar k, se produce un efecto de escalamiento sobre la longitud del vector por un factor de $|k|$ unidades, obteniendo al vector $k\mathbf{v}$ como resultado, el cual es paralelo a $\mathbf{v}$ y tiene una longitud $k\|\mathbf{v}\|$.

Aunado a esto, su dirección sigue siendo la misma sí $k > 0$, pero si $k < 0$ entonces resulta tener una dirección opuesta. Finalmente, sí $k = 0$, entonces $k\mathbf{v} = \mathbf{0}$ (Ver **Figura 2.4**)

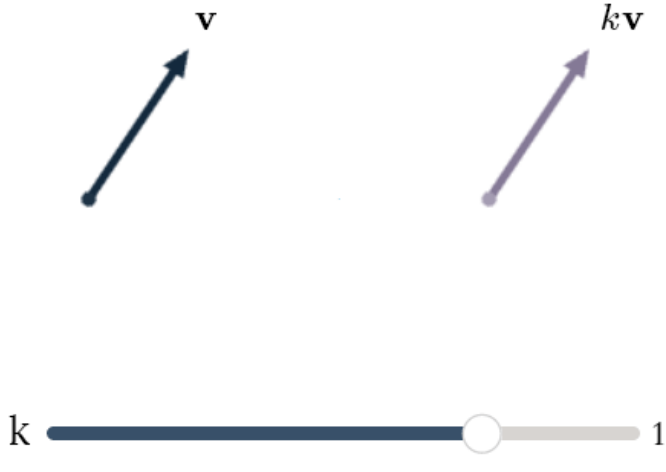


Figura 2.4. Se muestra como el vector $\mathbf{v}$ puede ser multiplicado por un vector k formando al vector $k\mathbf{v}$. Se invita al lector a modificar al vector $\mathbf{v}$ arrastrando la punta del vector a una posición diferente, así como cambiar el valor de k. Observe lo que ocurre con $k\mathbf{v}$ cuando $k < 0$.

2.3.2 Suma de vectores

La suma de dos vectores $\mathbf{u}$ y $\mathbf{v}$, se construye poniendo al vector $\mathbf{u}$ en alguna posición arbitraria, y al vector $\mathbf{v}$ de tal forma que su punto inicial coincida con el punto terminal o cabeza de $\mathbf{u}$. En la **Figura 2.5** se muestra dicha construcción, donde $\mathbf{u} + \mathbf{v}$ es la diagonal del paralelogramo formado por $\mathbf{u}$ y $\mathbf{v}$.

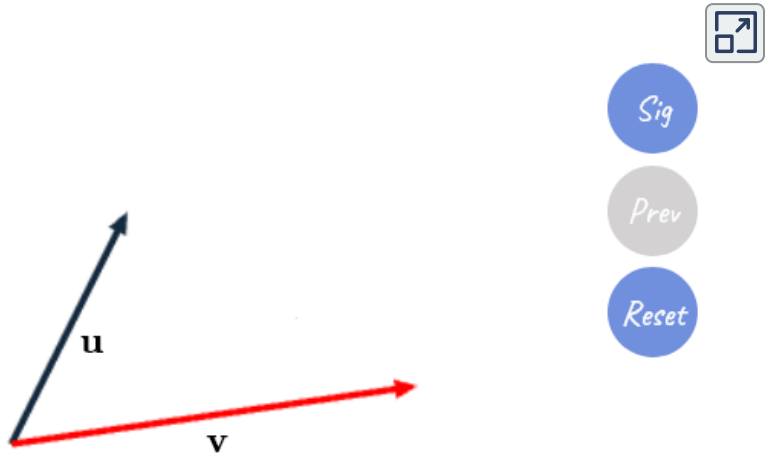


Figura 2.5. Suma de dos vectores 2D. El lector puede cambiar los vectores arrastrando la punta de sus flechas. Da clic en el botón *Sig* o *Prev* para ver el paso siguiente o previo.

Cuando se quiere sumar más de dos vectores, se construye la suma de manera análoga, colocando ahora un vector a continuación del otro (Ver **Figura 2.6**).

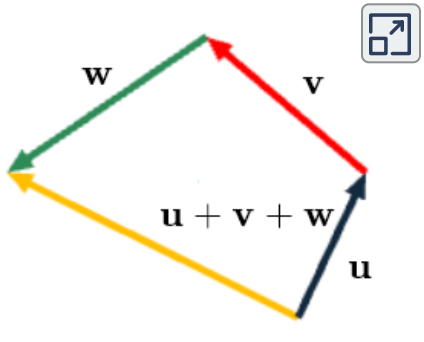


Figura 2.6. Suma de tres vectores $\mathbf{u}$, $\mathbf{v}$ y $\mathbf{w}$.

Hasta ahora solo hemos ilustrado la suma en dos dimensiones, sin embargo, las operaciones de los vectores también pueden ser ilustradas en tres dimensiones, como se muestra a continuación.

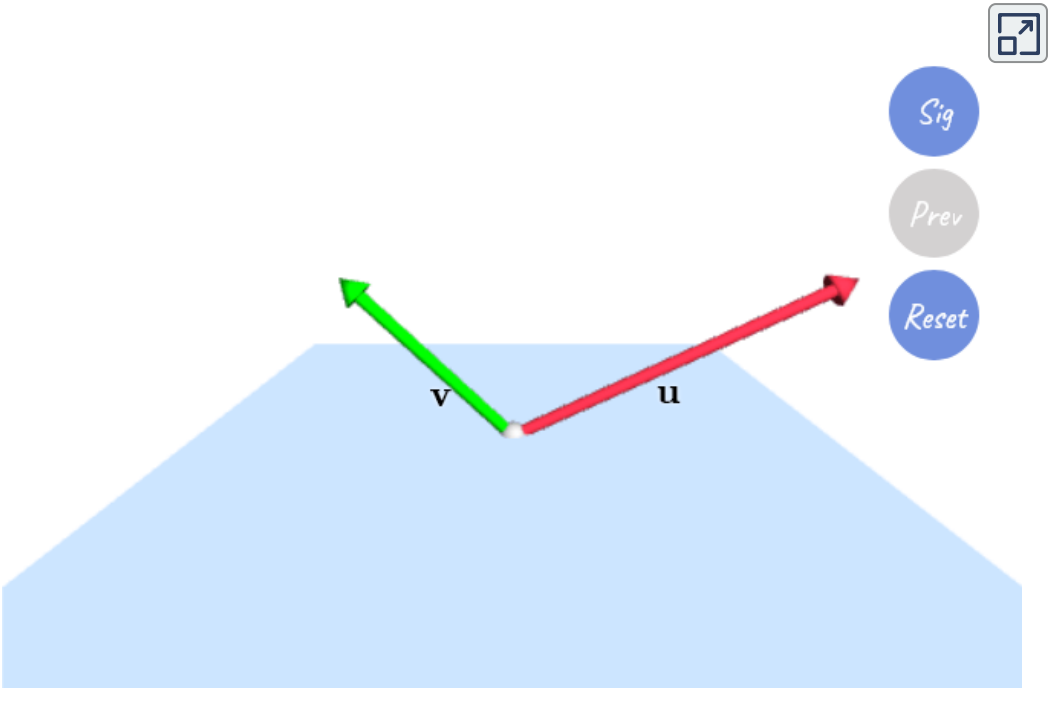


Figura 2.7. Suma de dos vectores 3D. Da clic en el botón *Sig* o *Prev* para ver el paso siguiente o previo. Se puede mover la posición de la cámara con el ratón o dedo.

Ahora bien, ya que hemos hablado de la suma y multiplicación por escalar de vectores, podemos seguir con la resta de vectores.

2.3.3 Resta de vectores

La resta de dos vectores $\mathbf{u} - \mathbf{v}$ es justamente una simplificación de

$$\underbrace{\mathbf{u} + \underbrace{(-1\mathbf{v})}_{\text{mult. por escalar}}}_{\text{suma}} = \mathbf{u} + (-\mathbf{v}) = \mathbf{u} - \mathbf{v}$$

(Ver **Figura 2.8**).

Donde $-\mathbf{v}$ se conoce como el negativo de $\mathbf{v}$ o su inverso aditivo.

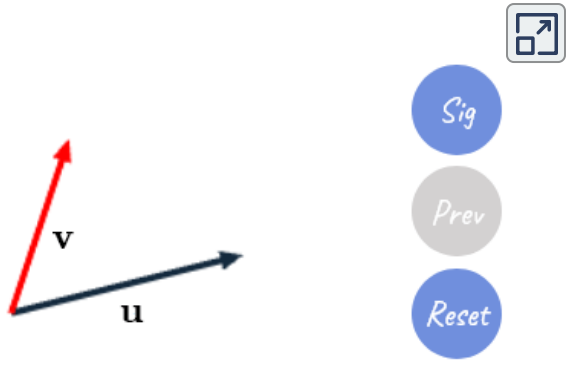


Figura 2.8. Resta de dos vectores 2D. Note que los vectores pueden cambiarse al arrastrar la punta de sus flechas. Da clic en el botón *Sig* o *Prev* para ver el paso siguiente o previo.

2.3.4 Espacios vectoriales, bases y coordenadas.

Un *espacio vectorial* V sobre un campo de escalares K se define como un conjunto (no vacío) de objetos, llamados vectores, con dos operaciones binarias: suma de vectores y multiplicación de vectores por un escalar; y que además satisfacen las siguientes propiedades:

Para cualesquiera tres vectores $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ en V, y dos escalares k y l en K:

(i)	$\mathbf{u}+\mathbf{v}=\mathbf{v}+\mathbf{u}$	(Conmutatividad)
(ii)	$(\mathbf{u}+\mathbf{v})+\mathbf{w}=\mathbf{v}+(\mathbf{u}+\mathbf{w})$	(Asociatividad)
(iii)	$\mathbf{v}+\mathbf{0}=\mathbf{v}$	(Existencia del vector cero o neutro aditivo)
(iv)	$\mathbf{v}+(-\mathbf{v})=\mathbf{0}$	(Existencia del negativo de un vector o inverso aditivo)
(v)	$(kl)\mathbf{v}=k(l\mathbf{v})$	(Asociatividad)
(vi)	$k(\mathbf{u}+\mathbf{v})=k\mathbf{v}+k\mathbf{u}$	(Distributividad 1)
(vii)	$(k+l)\mathbf{v}=k\mathbf{v}+l\mathbf{v}$	(Distributividad 2)
(viii)	$k\mathbf{0}=\mathbf{0}$	(Multiplicación por vector cero)

(ix) $0\mathbf{v} = \mathbf{0}$ (Multiplicación por escalar cero)

(x) $1\mathbf{v} = \mathbf{v}$ (Unidad escalar)

El campo al que nos limitaremos a usar será el conjunto de los números reales $\mathbb{R}$.

2.3.4.1 Bases y coordenadas

Los sistemas de coordenadas son utilizados para describir la posición de un objeto dentro de un espacio. El más usado es el sistema de coordenadas cartesianas, el cual consta de dos ejes perpendiculares del mismo tamaño, con el origen en el punto de intersección de éstos.

Podemos definir un sistema de coordenadas sobre cualquier espacio que queramos. Por ejemplo, sobre un mapa como en el siguiente interactivo.

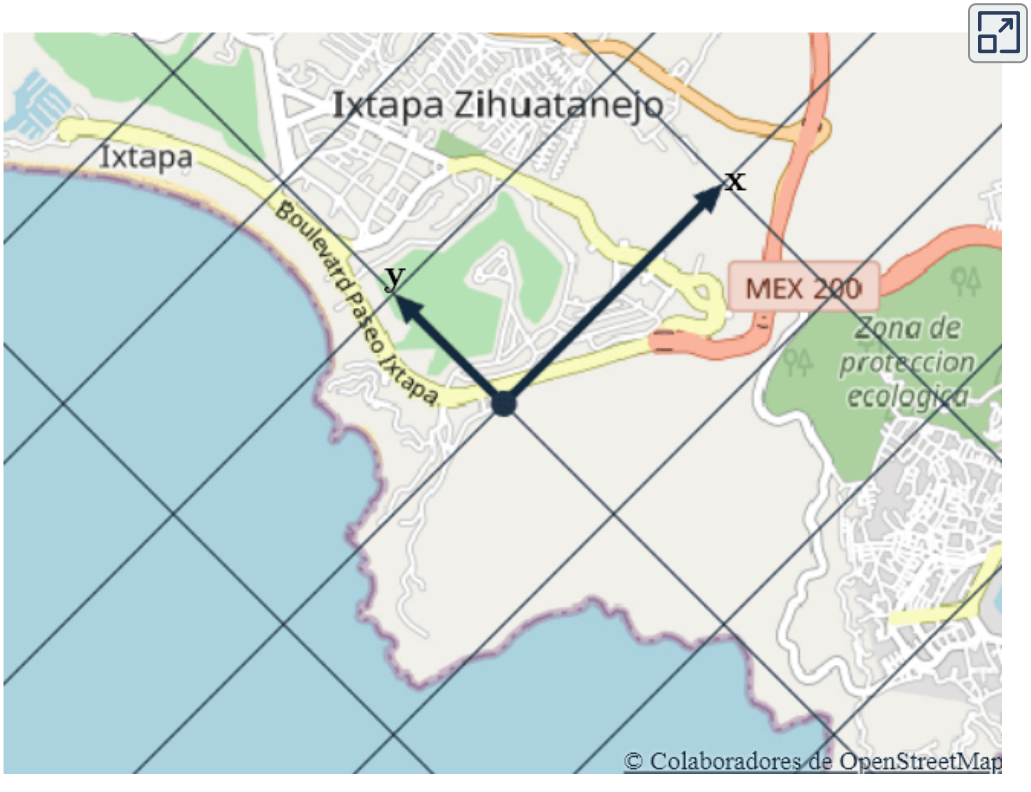


Figura 2.9. Mapa con sistema de coordenadas. El centro del mapa es utilizado como el origen o vector cero. Se muestran los ejes **x** y **y** del sistema como dos flechas, las cuales van variando de posición, al igual que las líneas de la cuadrícula, de modo que el sistema de coordenadas puede ser rotado o escalado y seguir funcionando como un sistema de referencia en el mapa.

Por otra parte, se dice que un espacio vectorial V es de *dimensión* $\boldsymbol{n}$ *finita* o $\boldsymbol{n}$*—dimensional* si tiene bases de n elementos. Por ejemplo, el sistema de coordenadas de la **Figura 2.9** es un espacio vectorial de dimensión 2, y se toman los vectores $\mathbf{x}$, y $\mathbf{y}$ como la base del espacio.

Podemos referirnos a un espacio vectorial n-dimensional sobre $\mathbb{R}$, simplemente como $\mathbb{R}^n$.

2.3.4.2 Sistemas de coordenadas unidimensional

Sea $\mathbf{e}$ un vector distinto al vector cero en un espacio unidimensional sobre $\mathbb{R}$, es decir, una línea recta. Se dice que $\mathbf{e}$ es un *vector base*, si para cualquier vector $\mathbf{v}$ en la línea, existe un único número x, tal que

$$\mathbf{v} = x\mathbf{e}$$

Siendo x la coordenada de $\mathbf{v}$ con $\{\mathbf{e}\}$ como base. Ver **Figura 2.10**.

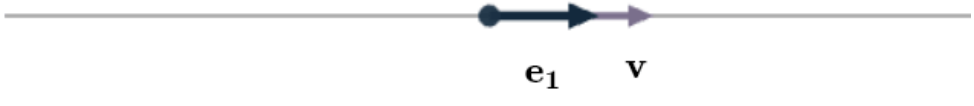


Figura 2.10. Sistema de coordenadas de dimensión 1. El vector $\mathbf{v}$ del interactivo puede moverse arrastrando la punta del vector.

2.3.4.3 Sistemas de coordenadas bidimensional

Sea $\mathbf{e_1}$ y $\mathbf{e_2}$ dos vectores sobre $\mathbb{R}^2$, distintos al vector cero y no paralelos entre sí. Se dice que $\mathbf{e_1}$ y $\mathbf{e_2}$ son *vectores base*, si para cualquier vector $\mathbf{v}$ en el plano, existe una única tupla (x, y), tal que

$$\mathbf{v} = x\mathbf{e_1} + y\mathbf{e_2}$$

Siendo x y y las coordenadas de $\mathbf{v}$ con $\{\mathbf{e_1}, \mathbf{e_2}\}$ como base. Ver **Figura 2.11**.

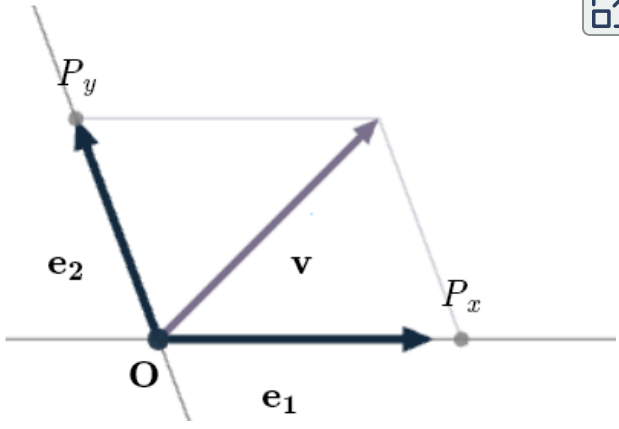


Figura 2.11. Sistema de coordenadas de dimensión 2. Nótese que $x\mathbf{e_1} + y\mathbf{e_2} = \mathbf{O}\vec{P}_x + \mathbf{O}\vec{P}_y = \mathbf{v}$. Todos los vectores pueden moverse arrastrando su punta.

2.3.4.4 Sistemas de coordenadas tridimensional

Sea $\mathbf{e_1}, \mathbf{e_2}$ y $\mathbf{e_3}$ tres vectores en $\mathbb{R}^3$ distintos del vector cero, con los cuales no se forma ningún plano paralelo entre todos ellos. Se dice que $\{\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}\}$ es una *base* , si para cualquier vector $\mathbf{v}$ en el espacio tridimensional, existe una única tripleta de coordenadas (x, y, z), tal que

$$\mathbf{v} = x\mathbf{e_1} + y\mathbf{e_2} + z\mathbf{e_3}$$

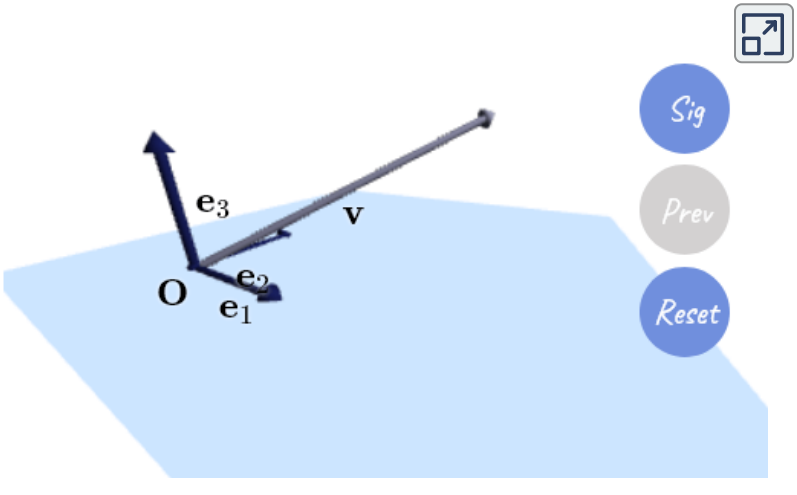


Figura 2.12. Sistema de coordenadas de dimensión 3. Se muestra cómo el vector $\mathbf{v}$ puede ser descrito con los vectores base $\mathbf{e_1}, \mathbf{e_2}$ y $\mathbf{e_3}$, utilizando las dos definiciones pasadas y la suma de vectores, (de clic en los botones). El lector puede mover y acercar o alejar la cámara con el ratón.

2.3.4.5 Sistemas de coordenadas n-dimensional

Análogamente, si tenemos al conjunto $S = \{e_1, e_2, ..., e_n\}$ como la *base* de un espacio $n - dimensional$ sobre $\mathbb{R}$, entonces, para cualquier vector $\mathbf{v}$ en $\mathbb{R}^n$, existe una única tupla de escalares (coordenadas) $(v_1, v_2, ..., v_n)$, tales que

$$\mathbf{v} = \sum_{i=1}^{n} v_i \mathbf{e_i}$$

y se dice que $\mathbf{v}$ es una *combinación lineal* de los vectores base.

Por otra parte, los vectores de una base siempre son *linealmente independientes* entre sí, esto es, que ninguno de los vectores puede ser expresado como múltiplo escalar de otro. De no ser así, se dice que el conjunto de vectores es *linealmente dependiente*.

Y ya que los vectores pueden ser identificados por sus coordenadas, es que surge la idea de representarlos por medio de las mismas.

Una forma de escribir a un vector es como un *vector columna*. Por ejemplo, si suponemos que la base que usaremos en un espacio tridimensional es $\{\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}\}$, entonces podemos escribir al vector $\mathbf{v}$ como

$$\mathbf{v} = v_x\mathbf{e_1} + v_y\mathbf{e_2} + v_z\mathbf{e_3} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

En este libro, los vectores serán denotados de está manera, o bien, de una manera más compacta como la $n - tupla$ de las n coordenadas del vector, que serán utilizados para representar puntos (posiciones) sobre los planos. En este mismo ejemplo esto sería $\mathbf{v} = (v_x, v_y, v_z)$.

Continuando, un *vector renglón* por otra parte, se escribe en horizontal

$$\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix},$$

siendo la *traspuesta* del vector columna, y viceversa.

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad ; \quad \mathbf{v}^T = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}$$

Podemos notar también que si partimos del punto de origen O dentro de un espacio vectorial, y nos movemos a un punto P dentro del mismo, entonces este desplazamiento puede ser representado por el vector $\vec{OP}$, que estaría descrito con las coordenadas de P.

Siguiendo esta idea es fácil ver que dados dos puntos en el plano P y Q, podemos obtener el vector que describe el desplazamiento de P a Q con la diferencia, es decir, $\vec{PQ} = Q - P$, pues nos interesa la distancia entre cada coordenada del punto de origen al punto destino.

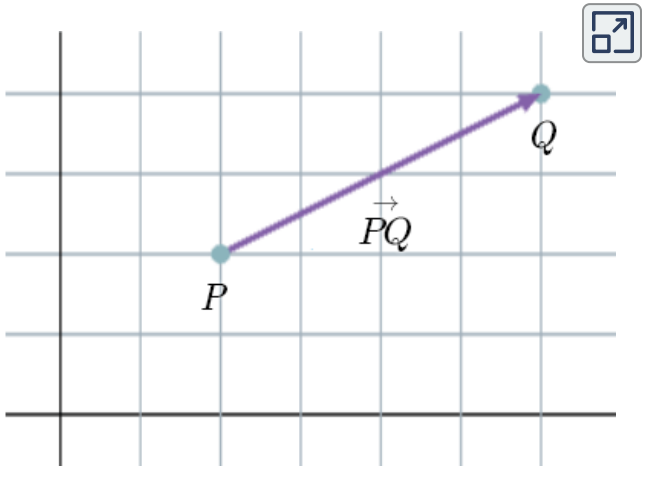


Figura 2.13. Visualización del vector generado del punto P al punto Q.

2.3.4.6 Multiplicación de vectores por un escalar y suma usando coordenadas

Sean dos vectores $\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$ y $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$ con la misma base, y un escalar k

la *multiplicación* $k\mathbf{v}$ está dada por

$$k\mathbf{v} = \begin{bmatrix} kv_1 \\ kv_2 \\ \vdots \\ kv_n \end{bmatrix}$$

Mientras que la *suma* $\mathbf{u} + \mathbf{v}$ se define como

$$\mathbf{u} + \mathbf{v} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

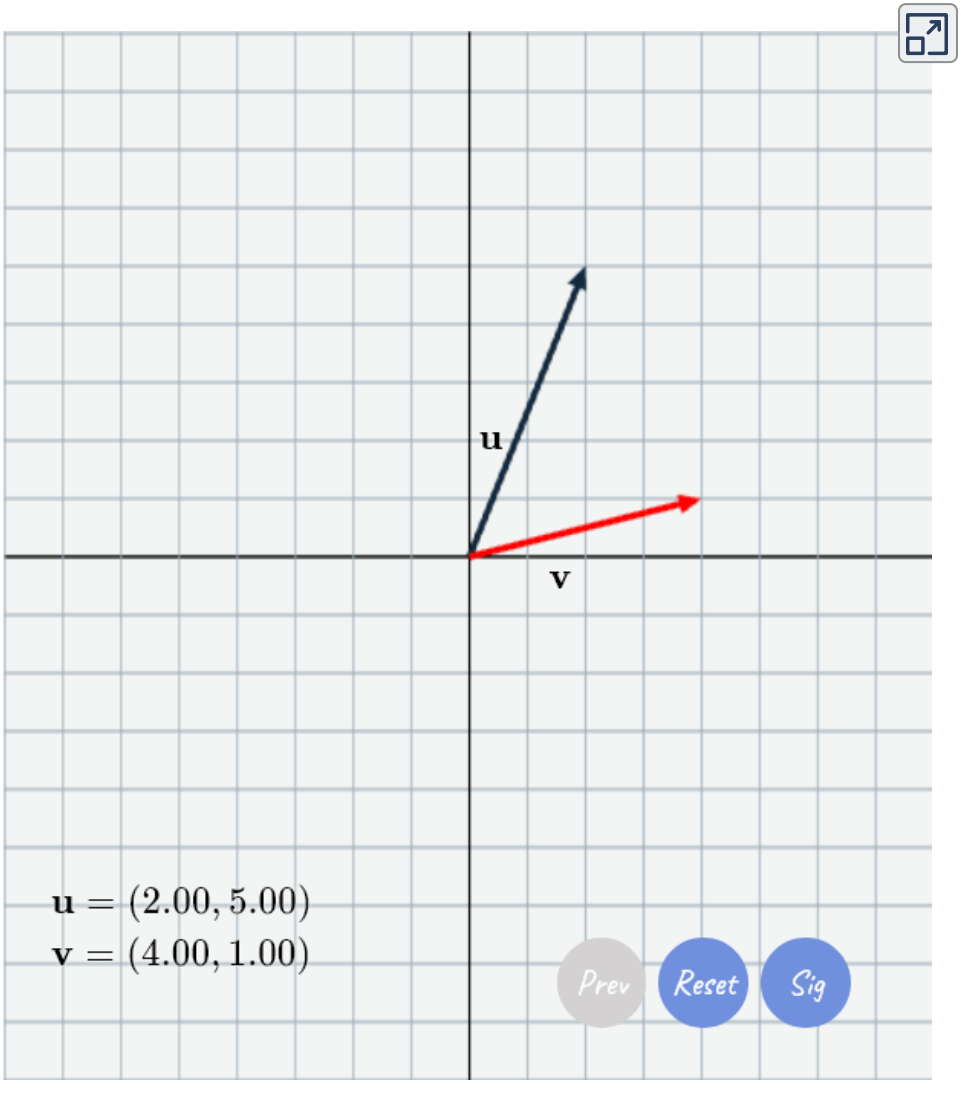


Figura 2.14. En el interactivo se expresa la suma de dos vectores sobre un plano 2D, utilizando la base estándar $\{(1,0),(0,1)\}$. De igual modo, se invita al lector a mover los vectores y observar el proceso dando clic en los botones *Sig* y *Prev* para ver cada paso.

Las *bases canónicas* o *estándar* suelen ser definidas como los vectores base e_i con todas sus entradas en 0, a excepción de su i-ésima componente, el cual es 1. Por ejemplo, la base estándar de un espacio 3D sería $\{(1,0,0),(0,1,0),(0,0,1)\}$

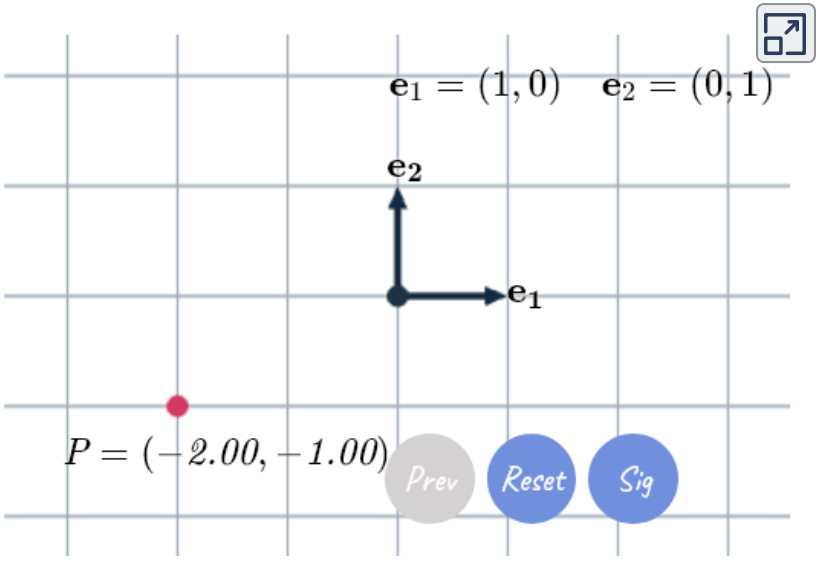


Figura 2.15. Se muestra el mismo punto expresado con diferentes bases, incluyendo la base estándar. Con los botónoes *Sig*, *Prev* y *Reset*, se puede cambiar de base. Nótese que las coordenadas del punto dependen de los vectores que se estén utilizando como base, por lo que cuando dichos vectores cambian, las coordenadas del punto también lo hacen, pero el punto permanece en su mismo lugar.

2.3.5 Producto punto

Primero, vale la pena recordar que dos vectores son *ortogonales*, si el ángulo más pequeño que se forma entre ellos, es de $\frac{\pi}{2}$ o 90°, siendo perpendiculares entre sí. Y que si el ángulo es igual a cero, los vectores son *colineales*, pues comparten la misma línea, siendo paralelos entre sí.

El *producto punto* de dos vectores $\mathbf{u}$ y $\mathbf{v}$, es denotado como $\mathbf{u} \cdot \mathbf{v}$, y también es conocido como producto escalar, ya que se define como el escalar

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$$

donde θ es el ángulo más pequeño comprendido entre los vectores $\mathbf{u}$ y $\mathbf{v}$.

Por consiguiente podemos deducir que:

$$\mathbf{u} \cdot \mathbf{v} = 0 \iff \mathbf{u} = \mathbf{0} \text{ o } \mathbf{v} = \mathbf{0}, \text{ o } \theta = \frac{\pi}{2} \text{ es decir, } \mathbf{u} \text{ y } \mathbf{v} \text{ son ortogonales.}$$

$$\mathbf{u} \cdot \mathbf{v} > 0 \iff 0 < \theta < \frac{\pi}{2}$$

$$\mathbf{u} \cdot \mathbf{v} < 0 \iff \frac{\pi}{2} < \theta \leq \pi$$

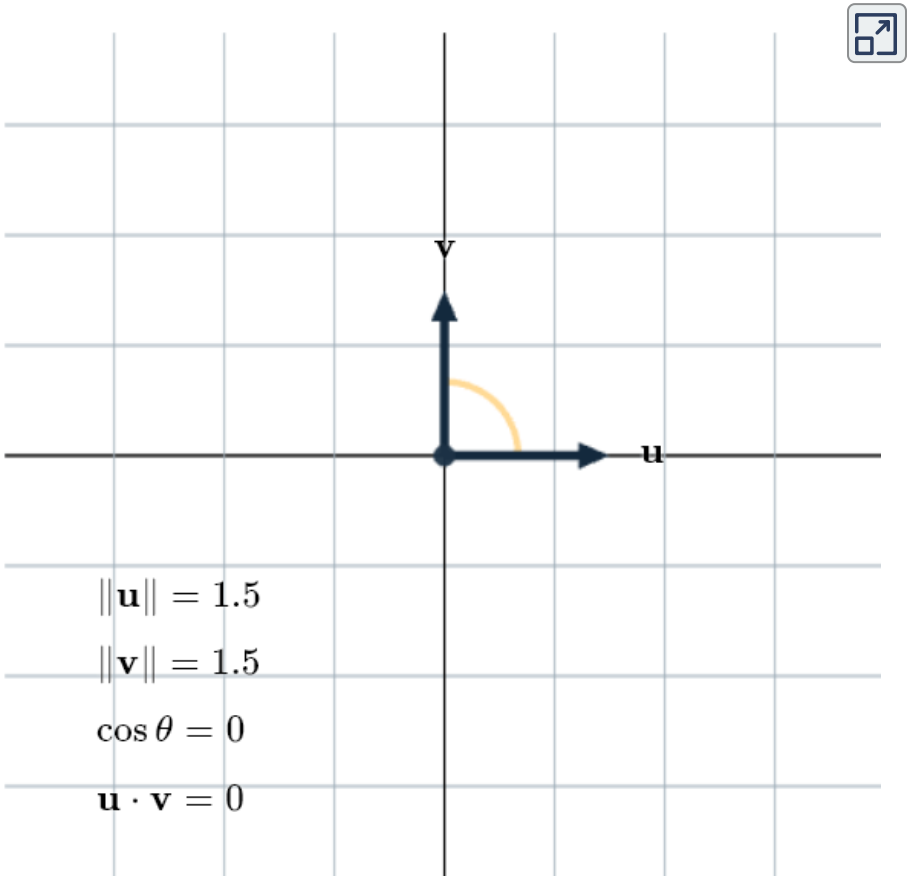


Figura 2.16. El producto punto entre dos vectores **u** y **v** en una base estándar, se invita al lector a mover los vectores y observar el efecto que tiene en los cálculos.

Un vector cuya magnitud es exactamente 1, es llamado *vector unitario*, esto es, $\mathbf{v}$ es vector unitario si $\|\mathbf{v}\| = 1$. Para cualquier vector $\mathbf{v}$ distinto al vector cero, se puede obtener su respectivo vector unitario, es decir, un vector $\mathbf{n}$ de longitud 1 con la misma dirección de $\mathbf{v}$.

Dicho proceso es conocido como *normalización* y se efectua cómo sigue

$$\mathbf{n} = \frac{1}{\|\mathbf{v}\|}\mathbf{v}$$

Por lo que, al tener dos vectores $\mathbf{v}$ y $\mathbf{u}$ con $\|\mathbf{v}\| = \|\mathbf{u}\| = 1$, el cálculo del producto punto se simplifica a $\mathbf{v} \cdot \mathbf{u} = \cos\theta$.

Siendo muy útil en GC, para el computo de los shaders, pues se suele requerir el coseno del ángulo entre dos vectores.

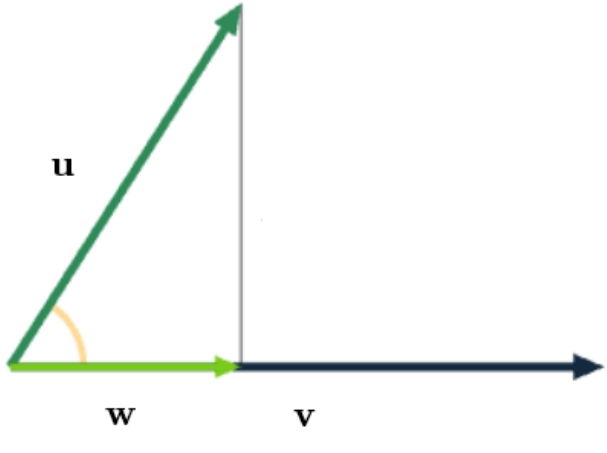
2.3.5.1 Proyección ortogonal

Supongamos que queremos proyectar *ortogonalmente* al vector $\mathbf{u}$ sobre el vector $\mathbf{v}$, obteniendo un nuevo vector $\mathbf{w}$ como la proyección resultante.

De trigonometría básica, sabemos que dentro de un triángulo rectángulo podemos relacionar al coseno de uno de los ángulos pequeños con la longitud de su respectivo cateto adyacente y la hipotenusa.

Como podemos notar en el siguiente interactivo, se forma un triángulo rectángulo. En este caso, podemos pensar al vector $\mathbf{u}$ como la hipotenusa, y al vector $\mathbf{w}$ como el cateto adyacente al ángulo θ que se forma entre los vectores $\mathbf{u}$ y $\mathbf{v}$.

Teniendo entonces $\cos\theta = \frac{\|\mathbf{w}\|}{\|\mathbf{u}\|}$.



Con ésto, podemos deducir que

$$\|\mathbf{w}\| = \|\mathbf{u}\| \cos\theta$$

De modo que para obtener al vector $\mathbf{w}$, simplemente multiplicamos al vector unitario de $\mathbf{v}$ por $\|\mathbf{w}\|$, es decir,

$$\mathbf{w} = \|\mathbf{w}\| \frac{\mathbf{v}}{\|\mathbf{v}\|} = \frac{\|\mathbf{u}\| \cos\theta}{\|\mathbf{v}\|} \mathbf{v}$$

Pues sabemos que la normalización no afecta la dirección del vector, y si multiplicamos por $\frac{\|\mathbf{v}\|}{\|\mathbf{v}\|}$ tenemos que

$$\mathbf{w} = \frac{\|\mathbf{u}\| \|\mathbf{v}\| \cos\theta}{\|\mathbf{v}\|^2} \mathbf{v}$$

$$\Rightarrow \mathbf{w} = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{v}\|^2} \mathbf{v}$$

Entonces si tenemos un vector $\mathbf{v}$ distinto al vector cero, podemos definir la *proyección de* $\mathbf{u}$ *sobre* $\mathbf{v}$ como

$$P_{\mathbf{v}} \mathbf{u} = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{v}\|^2} \mathbf{v}$$

Notemos que sí $\mathbf{v}$ es un vector unitario entonces la fórmula se reduce a solo $(\mathbf{u} \cdot \mathbf{v})\mathbf{v}$.

Propiedades del producto punto

A continuación se listan algunas propiedades útiles del producto punto. Sean $\mathbf{u}, \mathbf{v}$ y $\mathbf{w}$ tres vectores y k un escalar, se satisface lo siguiente

(i) $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$	(Conmutatividad)
(ii) $k(\mathbf{u} \cdot \mathbf{v}) = (k\mathbf{u}) \cdot \mathbf{v}$	(Asociatividad)
(iii) $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$	(Distributividad)
(iv) $\mathbf{v} \cdot \mathbf{v} = \|\mathbf{v}\|^2 \leq 0$	(Longitud cuadrática) Siendo cero únicamente cuando $\mathbf{v} = \mathbf{0}$

2.3.5.2 Cálculo del producto punto en una base ortonormal

Para cualquier base ortonormal n-dimensional $\{e_1, e_2, ..., e_n\}$ se satisface lo siguiente

$$e_i \cdot e_j = \begin{cases} 1 & \text{si } e_i = e_j \\ 0 & \text{si } e_i \neq e_j \end{cases}$$

y como $e_i \cdot e_i = 1$, quiere decir que la longitud de los vectores base debe ser uno, es decir, están normalizados y son ortogonales entre sí.

Entonces, en cualquier base ortonormal, el producto punto entre dos vectores n-dimensionales $\mathbf{u}$ y $\mathbf{v}$, está dado por

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

Por ejemplo, el producto punto en una base de dimensión 3 sería

$$\mathbf{u} \cdot \mathbf{v} = u_x v_x + u_y v_y + u_z v_z$$

2.3.5.3 Cálculo de la longitud de un vector en una base ortonormal

De la propiedad (iv) del producto punto tenemos que $\mathbf{v} \cdot \mathbf{v} = \|\mathbf{v}\|^2 \leq 0$, esto es, $\mathbf{v} \cdot \mathbf{v} = \|\mathbf{v}\|\|\mathbf{v}\| \cos\theta = \|\mathbf{v}\|^2$, pues $\theta = 0 \Rightarrow \cos\theta = 1$.

Entonces, si tomamos un vector 2D $\mathbf{v} = (v_x, v_y)$ en una base ortonormal, podemos utilizar (iv) para obtener su longitud, esto es $\|\mathbf{v}\|^2 = \mathbf{v} \cdot \mathbf{v} = v_x^2 + v_y^2$, por lo que

$$\|\mathbf{v}\| = \sqrt{v_x^2 + v_y^2}$$

La cual también es una prueba del teorema de Pitágoras (Ver **Figura 2.17**).

De igual manera, para un vector 3D $\mathbf{v} = (v_x, v_y, v_z)$ en una base ortonormal, su longitud está dada por

$$\|\mathbf{v}\| = \sqrt{v_x^2 + v_y^2 + v_z^2}.$$

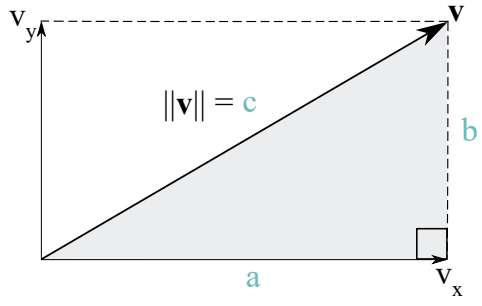


Figura 2.17. Relación de la longitud de un vector 2D con el teorema de Pitágoras.

Entonces, la *magnitud de un vector de una dimensión arbitraria* $\boldsymbol{n}$, en una base ortonormal, se calcula como

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2} = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}.$$

satisfaciendo también las siguientes propiedades para cualesquiera dos vectores $\mathbf{u}$ y $\mathbf{v}$ en V, y un escalar k en K:

(i) $\|\mathbf{v}\| \geq 0$

(ii) $\|\mathbf{v}\| = 0 \iff \mathbf{v} = \mathbf{0}$

(iii) Los vectores $\|k\mathbf{v}\| = |k|\|\mathbf{v}\|$

2.3.6 Producto cruz

El *producto cruz* de dos vectores tridimensionales, ya que solo está definido en $\mathbb{R}^3$, es definido como un nuevo vector $\mathbf{u} \times \mathbf{v}$ que cumple las siguientes propiedades:

(i) $\mathbf{u} \times \mathbf{v}$ es *ortogonal* a ambos vectores $\mathbf{u}$ y $\mathbf{v}$.

(ii) $\|\mathbf{u} \times \mathbf{v}\| = \|\mathbf{u}\|\|\mathbf{v}\| \sin\theta$, siendo θ el ángulo más pequeño comprendido entre los vectores $\mathbf{u}$ y $\mathbf{v}$.

(iii) Los vectores $\mathbf{u}$, $\mathbf{v}$ y $\mathbf{u} \times \mathbf{v}$ están orientados positivamente. (Ver **Figura 2.18**)

Entonces, por (ii) tenemos que $\mathbf{u} \times \mathbf{v} = \mathbf{0}$, si $\mathbf{u} = \mathbf{0}$ o $\mathbf{v} = \mathbf{0}$, Así también si el ángulo θ es igual a cero, es decir, si $\mathbf{u}$ y $\mathbf{v}$ son paralelos entre sí.

Por otro lado, un método muy utilizado para conocer la dirección de $\mathbf{u} \times \mathbf{v}$ es el de la *regla de la mano derecha*, donde se posiciona la mano derecha de modo que el dedo índice apunte a la misma dirección que el vector $\mathbf{u}$ y el dedo medio a la de $\mathbf{v}$, entonces, el pulgar apuntará en dirección del vector $\mathbf{u} \times \mathbf{v}$.

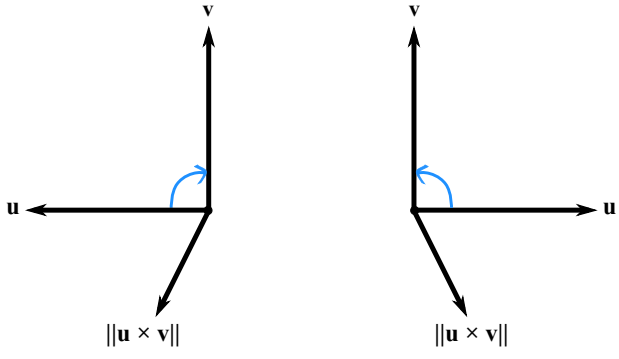
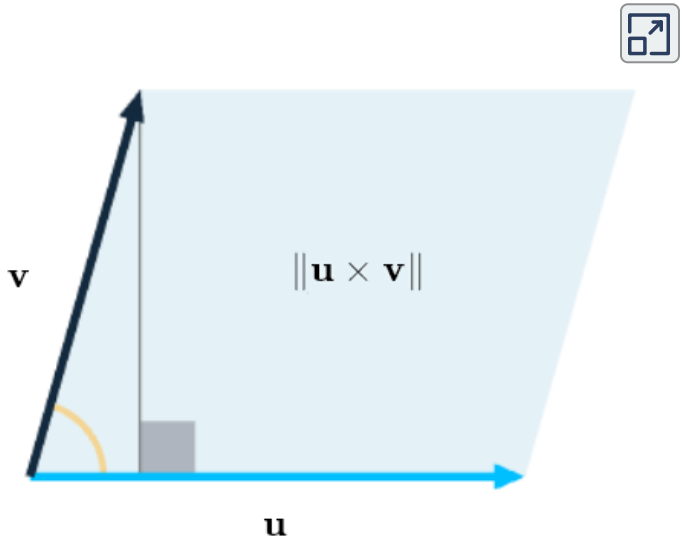


Figura 2.18. Los vectores $\mathbf{u}$, $\mathbf{v}$ y $\mathbf{u} \times \mathbf{v}$ posicionados a la izquierda se encuentran *orientados negativamente*, mientras que los de la derecha están *orientados positivamente*.

Notemos también que la longitud del producto cruz puede ser interpretada geométricamente como el área del paralelogramo formado por ambos vectores, pues $\|\mathbf{v}\| \sin\theta$ es la altura del triángulo formado por $\mathbf{u}$ y $\mathbf{v}$, que es también la altura del paralelogramo, entonces

$$a = bh = \|\mathbf{u}\|\|\mathbf{v}\| \sin\theta = \|\mathbf{u} \times \mathbf{v}\|$$



El cálculo del producto cruz suele ser muy utilizado para gráficos 3D en particular, la propiedad (i) nos permite calcular la normal de una superficie en un punto en específico al utilizar dos vectores tangentes.

$$\mathbf{u} = (0, 0, 1)$$
$$\mathbf{v} = (1, 0, 0)$$

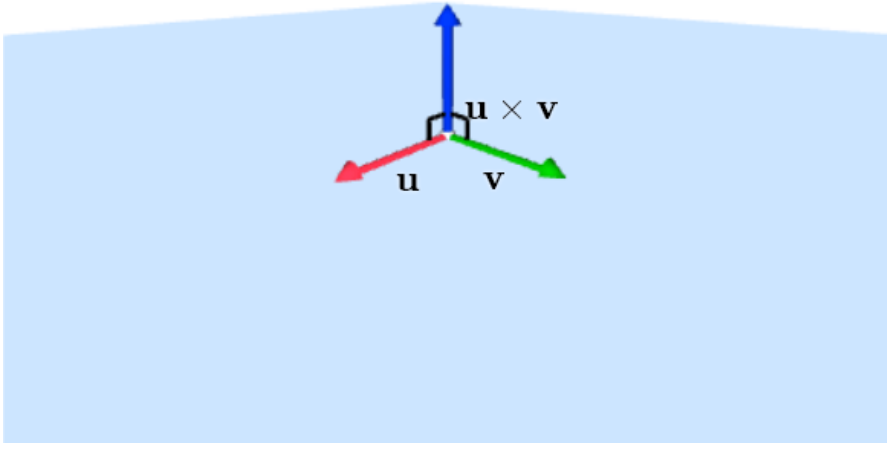


Figura 2.19. Producto cruz. El interactivo muestra el producto cruz $\mathbf{u} \times \mathbf{v}$ entre dos vectores. Se invita al lector a modificar a los vectores cambiando las entradas de sus coordenadas y observar lo que ocurre cuando se hace a $\mathbf{u}$ y $\mathbf{v}$ paralelos, o cambian de orden. La cámara puede moverse con el ratón o dedo y acercarse o alejarse con la rueda del ratón.

Propiedades del producto cruz

(i) $\mathbf{u} \times \mathbf{v} = -\mathbf{v} \times \mathbf{u}$ (Conmutatividad)

(ii) $\mathbf{u} \times (\mathbf{v} + \mathbf{w}) = \mathbf{u} \times \mathbf{v} + \mathbf{u} \times \mathbf{w}$ (Distributividad)

(iii) $(\mathbf{u} + \mathbf{v}) \times \mathbf{w} = \mathbf{u} \times \mathbf{w} + \mathbf{v} \times \mathbf{w}$ (Distributividad)

(iv) $k(\mathbf{u} \times \mathbf{v}) = (k\mathbf{u}) \times \mathbf{v} = \mathbf{u} \times (k\mathbf{v})$ (Asociatividad)

2.3.6.1 Producto cruz en base ortonormal

Sean dos vectores $\mathbf{u}$ y $\mathbf{v}$ en $\mathbb{R}^3$, el producto cruz está dado por

$$\mathbf{u} \times \mathbf{v} = (u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_x)$$

Una forma de recordar esta fórmula es utilizando el *pseudodeterminante*

$$\mathbf{u} \times \mathbf{v} = \begin{vmatrix} e_1 & e_2 & e_3 \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} = e_1 \begin{vmatrix} u_y & u_z \\ v_y & v_z \end{vmatrix} - e_2 \begin{vmatrix} u_x & u_z \\ v_x & v_z \end{vmatrix} + e_3 \begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix}$$

$$= (u_y v_z - u_z v_y)e_1 + (u_z v_x - u_x v_z)e_2 + (u_x v_y - u_y v_x)e_3$$

Se dice que es pseudodeterminante ya que el primer renglón consta de vectores, considerando que las entradas deberían de ser solo escalares.

Otra forma de expresar el producto cruz es con la transformación lineal

$$\mathbf{u} \times \mathbf{v} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Los temas de determinantes, matrices y transformaciones lineales serán tratados con más detenimiento en los siguientes subtemas.

2.4 Matrices

Las matrices son una herramienta muy poderosa para manipular datos, y es por esta razón que son fundamentales en GC.

Una *matriz* $\mathbf{A}$ de dimensión $r \times c$ se define como un arreglo rectangular de rc escalares a_{ij} distribuidos en un orden de r reglones y c columnas, representada de la siguiente forma:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1c} \\ a_{21} & a_{22} & \cdots & a_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \cdots & a_{rc} \end{bmatrix}$$

Siendo a_{ij} el número que aparece en el renglón i y en la columna j, también llamado como $ij - ésima$ componente o elemento de $\mathbf{A}$.

Notemos que en una matriz de $r \times c$ hay r diferentes vectores renglón y c diferentes vectores columna, los primeros compuestos por c escalares y los segundos por r.

Y a su vez, una matriz con una sola columna es una matriz de dimensión $r \times 1$ o vector columna, asimismo, para una matriz con un solo renglón, se dice que es una matriz de dimensión $1 \times c$ o vector renglón.

Por otro lado, si $\mathbf{A}$ es una matriz con $r = c$ se dice que es una *matriz cuadrada*, y éstas son el tipo de matrices con las que se suele trabajar en GC.

Una *matriz identidad* $\mathbf{I}$ es una matriz cuadrada cuyos elementos de la *diagonal principal* son iguales a 1 y todos los demás son 0. Por ejemplo la matriz identidad de 3×3 sería

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.4.1 Operaciones de las matrices

2.4.1.1 Multiplicación por escalar

Se puede multiplicar una matriz $\mathbf{A}$ por un escalar k, obteniendo una matriz $S = k\mathbf{A}$ con las mismas dimensiones que $\mathbf{A}$, donde cada elemento está dado por ka_{ij}.

$$k\mathbf{A} = \begin{bmatrix} ka_{11} & ka_{12} & \cdots & ka_{1c} \\ ka_{21} & ka_{22} & \cdots & ka_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ ka_{r1} & ka_{r2} & \cdots & ka_{rc} \end{bmatrix}$$

2.4.1.2 Suma de matrices

Si dos matrices $\mathbf{A}$ y $\mathbf{B}$ tienen las mismas dimensiones, entonces pueden sumarse, obteniendo una nueva matriz $\mathbf{A} + \mathbf{B}$, donde cada elemento está dado por $a_{ij} + b_{ij}$.

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1c} + b_{1c} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2c} + b_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} + b_{r1} & a_{r2} + b_{r2} & \cdots & a_{rc} + b_{rc} \end{bmatrix}$$

2.4.1.3 Resta de matrices

Con las dos operaciones previas podemos definir la resta de dos matrices con la misma dimensión como

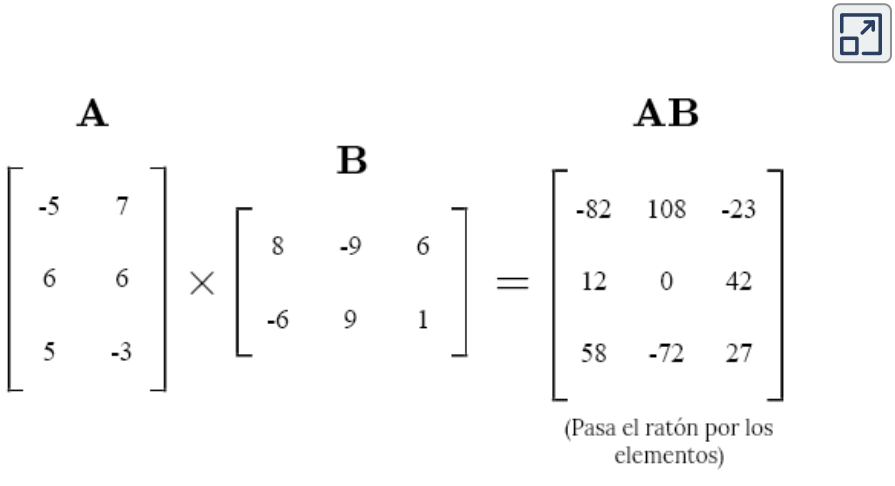
$$\mathbf{A} - \mathbf{B} = \mathbf{A} + (-1)\mathbf{B}$$

2.4.1.4 Multiplicación de matrices

Sí $\mathbf{A}$ es una matriz de $r \times s$ y $\mathbf{B}$ una matriz de $s \times t$, es decir, el número de columnas de la primera matriz es el mismo que el número de renglones que la segunda, entonces se pueden multiplicar, obteniendo la matriz $\mathbf{P} = \mathbf{AB}$ de tamaño $r \times t$, donde cada elemento

$$p_{ij} = \sum_{k=1}^{s} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + ... + a_{is} b_{sj}$$

Por ejemplo, al multiplicar una matrix $\mathbf{A}$ de 3×2 por una matriz $\mathbf{B}$ de 2×3



$$ab_{ij} = \sum_{k=1}^{2} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j}$$

Otro

Teniendo como resultado una matriz de dimensión $(3 \times \cancel{2})(\cancel{2} \times 3) = 3 \times 3$.

Nótese que al utilizar matrices cuadradas de orden n podemos realizar todas estas operaciones sin ningún problema, obteniendo nuevamente una matriz de $n \times n$ como resultado.

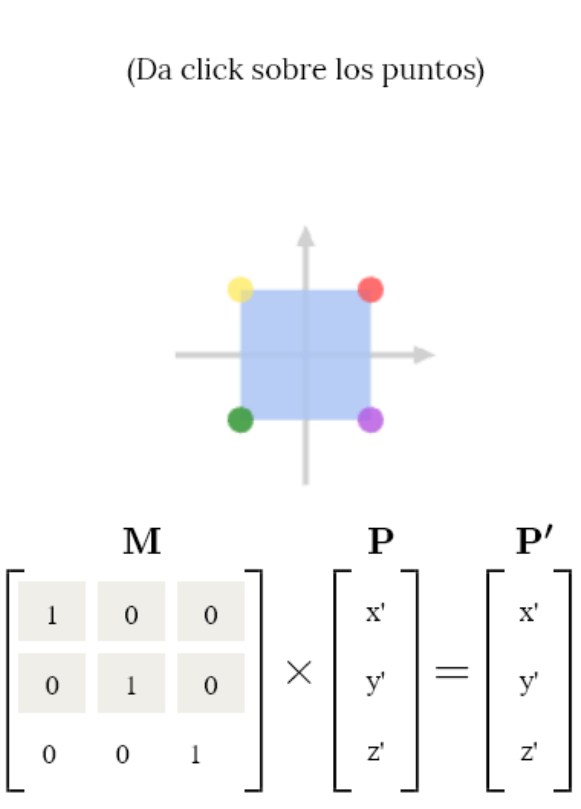


Figura 2.20. Manipulación de puntos con una matriz. Se pretende que el alumno modifique las entradas de la matriz y vea su efecto sobre los vértices del cuadro azul.

La *matriz inversa* de $\mathbf{A}$ se denota como $\mathbf{A^{-1}}$ y es aquella que nos asegura $\mathbf{AA^{-1}} = \mathbf{A^{-1}A} = \mathbf{I}$. No todas las matrices son invertibles.

En particular, una matriz cuadrada que no es invertible se denomina *singular*. Por otra parte, la inversa del producto de dos matrices es el producto de las inversas en orden contrario

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$$

La *traspuesta* de una matriz $\mathbf{A}$ de $r \times c$ se denota como $\mathbf{A}^T$, y es la matriz de $c \times r$ que se obtiene de intercambiar los renglones por columnas, es decir, donde cada entrada a_{ij} de A corresponde a la entrada a'_{ji} de A^T.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1c} \\ a_{21} & a_{22} & \cdots & a_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \cdots & a_{rc} \end{bmatrix}; \mathbf{A^T} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{r1} \\ a_{12} & a_{22} & \cdots & a_{r2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1c} & a_{2c} & \cdots & a_{rc} \end{bmatrix}$$



$$\underset{\mathbf{A}}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}} \quad \underset{\mathbf{A^T}}{\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}}$$

En otras palabras, el renglón i de $\mathbf{A}$ se escribe como la columna i de $\mathbf{A^T}$, y la columna j de $\mathbf{A}$ como el renglón j de $\mathbf{A^T}$.

Y visto que un vector puede ser representado como una matriz de una sola columna (o renglón), entonces podemos obtener la multiplicación de dos matrices, o bien, vectores, con el producto punto, pues

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T\mathbf{v} = \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \sum_{i=1}^{n} u_i v_i$$

Además, se dice que una matriz $\mathbf{A}$ es *simétrica* si es una matriz *cuadrada* y se tiene que $\mathbf{A} = \mathbf{A}^T$.

Del manera similar que con la inversa del producto de dos matrices pasa con la traspuesta, es decir

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$$

Por último, un conjunto especial de matrices son las llamadas *matrices ortogonales*, donde una *matriz ortogonal* $\mathbf{B}$ es una matriz cuadrada conformada por vectores columna (o renglón) los cuales constituyen una base ortonormal. Por lo que se satisface que

$$\mathbf{B^{-1}} = \mathbf{B^T} \quad \text{pues} \quad \mathbf{BB^T} = \mathbf{I}$$

ya que por definición de base ortonormal tenemos que $b_i \cdot b_i = 1$ y $b_i \cdot b_j = 0$, sí $i \neq j$.

Esta propiedad suele ser muy conveniente en GC ya que se suelen usar frecuentemente este tipo de matrices, por ejemplo, la matriz de *rotación* descrita en el Capítulo 4, o bien, como el uso de bases ortogonales para realizar cambios de base, lo cual se explica más adelante.

Se tiene también que $\|\mathbf{Bv}\| = \|\mathbf{v}\|$ y de ahí que $(\mathbf{Bu}) \cdot (\mathbf{Bv}) = \mathbf{u} \cdot \mathbf{v}$. Finalmente, si multiplicamos dos matrices ortogonales $\mathbf{A}$ y $\mathbf{B}$, entonces $\mathbf{AB}$ también será ortogonal.

Propiedades de las matrices

(i)	$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$	(Conmutatividad)
(ii)	$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$	(Asociatividad)
(iii)	$\mathbf{IA} = \mathbf{A}$	(Identidad multiplicativa)
(iv)	$(kl)\mathbf{A} = k(l\mathbf{A})$	(Asociatividad)
(v)	$k(\mathbf{A} + \mathbf{B}) = k\mathbf{A} + k\mathbf{B}$	(Distributividad)
(vi)	$(k + l)\mathbf{A} = k\mathbf{A} + l\mathbf{A}$	(Distributividad)
(vii)	$\mathbf{A} + (-1)\mathbf{A} = \mathbf{O}$	(Inverso aditivo. Nótese que la matriz de la derecha se trata de la matriz cero)
(viii)	$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$	(Distributividad)
(ix)	$(\mathbf{A} + \mathbf{B})\mathbf{C} = \mathbf{AC} + \mathbf{BC}$	(Distributividad)
(x)	$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$	(Asociatividad)
(xi)	$(k\mathbf{A})^T = k(\mathbf{A})^T$	(Traspuesta 1)
(xii)	$(\mathbf{A}^T)^T = \mathbf{A}$	(Traspuesta 2)
(xiii)	$(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$	(Traspuesta 3)
(xiv)	$(\mathbf{AB})^T = \mathbf{BA}$	(Traspuesta 4)

2.4.2 Sistemas de ecuaciones lineales

Las matrices nos proporcionan una forma compacta de expresar sistemas de ecuaciones lineales, por ejemplo, el siguiente sistema

$$\begin{cases} 2x_1 + 4x_2 + 6x_3 = 18 \\ 4x_1 + 5x_2 + 6x_3 = 24 \\ 3x_1 + x_2 - 2x_3 = 4 \end{cases}$$

puede ser representado como

$$\underbrace{\begin{bmatrix} 2 & 4 & 6 \\ 4 & 5 & 6 \\ 3 & 1 & -2 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 18 \\ 24 \\ 4 \end{bmatrix}}_{b}$$

$$\Longleftrightarrow$$

$$\mathbf{Ax} = \mathbf{b}$$

donde cada columna de $\mathbf{A}$ corresponde a los coeficientes de las variables del sistema, $\mathbf{x}$ el vector columna con las variables o incógnitas y $\mathbf{b}$ el vector columna de los términos independientes.

Existen varias formas de resolver este tipo de sistemas, por mencionar algunos, el método de *eliminación Gausiana* el cual es uno de los más conocidos y la *regla de Cramer*, el cual suele ser un método apropiado ya que en GC se suele trabajar con matrices cuadradas de dimensión menor o igual a 4.

En el método de *eliminación de Gaussiana* se obtiene las soluciones mediante la reducción del sistema de ecuaciones dado utilizando *operaciones elementales*, dichas operaciones se aplican a los renglones de la *matriz aumentada*, transformando la matriz de coeficientes en una matriz triangular superior (o sistema triangular) para así realizar las sustituciones adecuadas y obtener las soluciones.

Las operaciones elementales son las siguientes:

- $R_i \leftrightarrow R_j$ Intercambiar dos renglones R_i y R_j.
- $R_i \rightarrow kR_i$ Reemplazar el i-ésimo renglón por un múltiplo de ese mismo no nulo.

- $R_i \to R_i + kR_j$ Reemplazar el i-ésimo renglón por ese mismo más un múltiplo del j-ésimo renglón.

Por ejemplo, utilizando el sistema de ecuaciones anterior su matriz aumentada sería

$$\left[\begin{array}{ccc|c} 2 & 4 & 6 & 18 \\ 4 & 5 & 6 & 24 \\ 3 & 1 & -2 & 4 \end{array}\right]$$

Y utilizando las operaciones elementales, podemos resolver el sistema de la siguiente forma

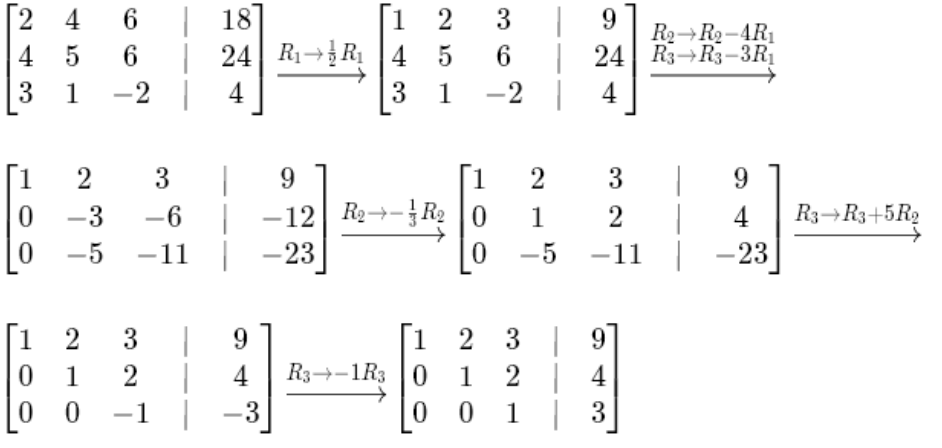


Figura 2.21. Ejemplo del desarrollo del método de eliminación gaussiana, en el cual puede verse cada paso utilizando los botónes *Sig*, *Prev* y *Reset (para volver al primer paso)*.

Mientras que con *la regla de Cramer* se calcula el valor de cada incógnita utilizando determinates. Es decir, considerado el sistema de ecuaciones $\mathbf{Ax} = \mathbf{b}$, podemos obtener las soluciones con

$$x_j = \frac{\det(\mathbf{A}_j)}{\det(\mathbf{A})}$$

donde $\mathbf{A}_j$ es la matriz resultante de reemplazar la j-ésima columna de $\mathbf{A}$ por el vector columna $\mathbf{b}$. Vale la pena mencionar que el sistema tendrá una solución única si y solo si $\det(\mathbf{A}) \neq 0$.

Entonces la solución del sistema de ecuaciones anterior estaría dada por

$$x_1 = \frac{\begin{vmatrix} 18 & 4 & 6 \\ 24 & 5 & 6 \\ 4 & 1 & -2 \end{vmatrix}}{\begin{vmatrix} 2 & 4 & 6 \\ 4 & 5 & 6 \\ 3 & 1 & -2 \end{vmatrix}} \quad x_2 = \frac{\begin{vmatrix} 2 & 18 & 6 \\ 4 & 24 & 6 \\ 3 & 4 & -2 \end{vmatrix}}{\begin{vmatrix} 2 & 5 & 6 \\ 4 & 5 & 6 \\ 3 & 1 & -2 \end{vmatrix}} \quad x_3 = \frac{\begin{vmatrix} 2 & 4 & 18 \\ 4 & 5 & 24 \\ 3 & 1 & 4 \end{vmatrix}}{\begin{vmatrix} 2 & 4 & 6 \\ 4 & 5 & 6 \\ 3 & 1 & -2 \end{vmatrix}}$$

Los sistemas lineales como el que hemos visto, es decir, en los que su vector $\mathbf{b}$ es distinto de cero se le denomina *no homogéneos*. Para este tipo de sistemas, existen tres posibilidades, que tenga:

- una solución única
- no tenga solución o
- un número infinito de soluciones.

Por el contrario, cuando todas las constantes que constituyen a su vector $\mathbf{b}$ son iguales a cero, se le denominan *homogéneos*. Y son un caso especial ya que solo tiene la posibilidad de tener una única solución, que es la *trivial*, donde todas las variables son cero, o bien, tiene un número infinito de soluciones.

En la **Figura 2.22** se observa que se puede describir geométricamente un sistema de dos ecuaciones con dos incógnitas mediante dos líneas rectas en $\mathbb{R}^2$. Si las líneas tienen un punto de intersección entonces tiene una solución única; si coinciden tienen un número infinito de soluciones y si son paralelas no se tiene ninguna solución. Algo similar se puede observar en el segundo interactivo (**Figura 2.23**), donde cada ecuación corresponde a un plano en $\mathbb{R}^3$.

Sistemas de ecuaciones con 2 incógnitas

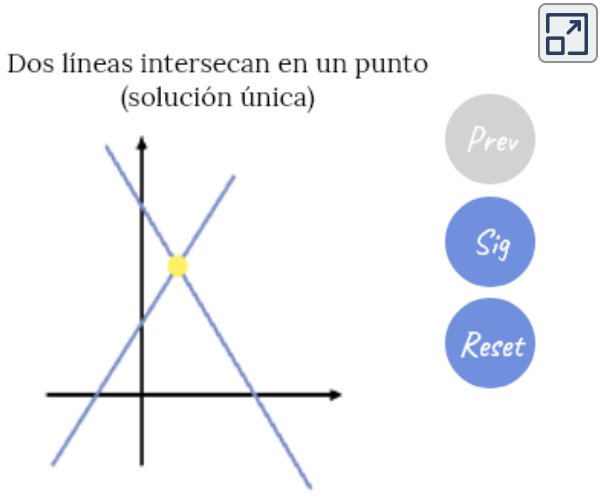


Figura 2.22. Representación geométrica de un sistema de dos ecuaciones con dos incógnitas.

Sistemas de ecuaiones con 3 incógnitas

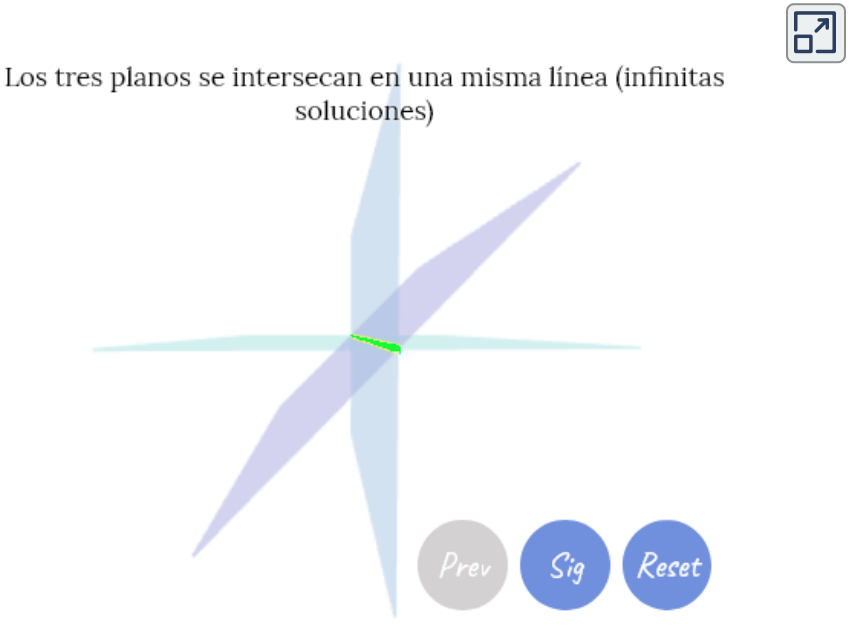


Figura 2.23. Representación geométrica de un sistema de tres ecuaciones con tres incógnitas. Se invita al lector a mover la posición de la cámara usando el ratón.

2.4.3 Determinantes

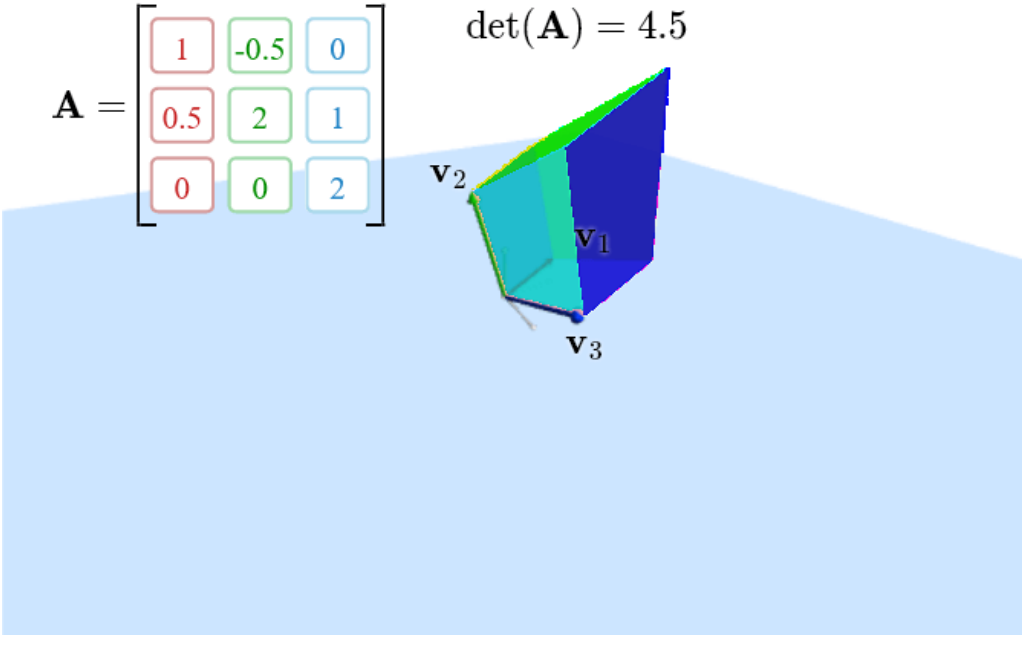


Figura 2.24. Paralelepípedo. Se visualizan los vectores columna de la matriz $\mathbf{A}$: $\mathbf{v}_1$, $\mathbf{v}_2$ y $\mathbf{v}_3$, los cuales se invita a modificarlos, y observar la forma del volumen del paralelepípedo resultante. También se puede mover la cámara usando el ratón, y acercarla o alejarla con la rueda del ratón.

El *determinante* de una matriz n-cuadrada $\mathbf{A}$ es un escalar especial derivado de las entradas de dicha matriz y se denota como $\det(\mathbf{A})$ o

$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

El determinante para matrices de 1 y 2 dimensiones

$$\left|a_{11}\right| = a_{11} \quad \text{y} \quad \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

Y si pensamos las dos columnas de la matriz como dos vectores columna en el plano, como vimos en el producto cruz, el determinante es el área (con signo) del paralelogramo formado por dichos vectores.

Consideremos ahora una matriz de dimensión 3, el determinante está dado por

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$= a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}$$

Y análogamnte si pensamos las tres columnas de la matriz como tres vectores columna el determinante nos da el volumen del paralelepípedo formado por los tres vectores (interactivo de la página anterior).

Como podemos notar el determinate está dado por una fórmula recursiva. De hecho, el *desarrollo de Laplace*, el cual nos permite calcular el determinante de matrices de elevadas dimensiones, consiste en descomponer al determinante en una suma de determinates menores, veamos cómo.

Consideremos a la matriz $\mathbf{A}$ de $n \times n$ y a la matriz $\mathbf{A}_{ij}$ de $(n-1) \times (n-1)$ que se obtiene de $\mathbf{A}$ al eliminar el i-ésimo renglón y la j-ésima columna, llamada *menor* ij de $\mathbf{A}$.

Ahora bien, definamos al *cofactor* ij de $\mathbf{A}$ como

$$C(\mathbf{A})_{ij} = (-1)^{i+j} \det(\mathbf{A}_{ij})$$

es decir, se toma al determinate del menor ij y multiplicándolo por $(-1)^{i+j}$. Observese que $(-1)^{i+j}$ es 1 cuando $i+j$ es par, y -1 en otro caso.

Entonces el *determinante de* $\mathbf{A}$ se calcula como

$$\det(\mathbf{A}) = \sum_{k=1}^{n} a_{ik} C(\mathbf{A})_{ik} \text{ sobre el renglón } i$$

O bien como

$$\sum_{k=1}^{n} a_{kj} C(\mathbf{A})_{kj} \text{ sobre la colúmna } j$$

Podemos ver este proceso, en particular, sobre el primer renglón, en el cálculo de los determinates de dos y tres dimensiónes descritos previamente. En términos de la fórmula, esto es

$$\sum_{k=1}^{n} a_{1k} C(\mathbf{A})_{1k} = \sum_{k=1}^{n} (-1)^{1+k} a_{1k} \det(\mathbf{A}_{1k}).$$

Como podemos notar, a medida que n incremente, los cálculos se vuelven astronómicos y el algoritmo ineficiente. No obstante, se suelen utilizan algunas de las propiedades que se enlistan a continuación para reducir los cálculos. Por otra parte, como mencionábamos antes, esto no suele ser un problema en graficación porque las matrices que se utilizan tienen en su mayoría no más de cuatro dimensiones.

Propiedades de los determinantes

(i) $\det(\mathbf{I}) = 1$

(ii) $|\cdots \mathbf{a}_i \ \ldots \mathbf{a}_j \ \ldots| = 0$ — es cero si dos columnas son múltiplos

(iii) $|\ldots k_1 \, \mathbf{a}_1 \ + k_2 \, \mathbf{a}_2 \ \ldots|$
$= k_1 \, |\ldots \mathbf{a}_1 \ \ldots| + k_2 \, |\ldots \mathbf{a}_2 \ \ldots|$

(iv) $|\cdots \mathbf{0} \ldots| = 0$ — es cero si alguna columna es cero $\mathbf{0}$

(v) $|\ldots \mathbf{a}_i \ \ldots \mathbf{a}_j \ \ldots| = -|\ldots \mathbf{a}_j \ \ldots \mathbf{a}_i \ \ldots|$ — intercambio de columnas (o renglones)

(vi) $|\cdots \mathbf{a}_i \ \ldots \mathbf{a}_j \ \ldots| = |\cdots \mathbf{a}_i \ + k\mathbf{a}_j \ \ldots \mathbf{a}_j \ \ldots|$ — suma del múltiplo de una columna a otra

(vii) $\det(\mathbf{A}) = \det(\mathbf{A}^T)$

(vii) $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$

(ix) $\det(\mathbf{A}^{-1}) = \frac{1}{\det(\mathbf{A})}$

2.4.3.1 Cáculo de la matriz inversa

Para calcular la matriz inversa primero necesitamos definir a la matriz *adjunta* de una matriz cuadrada $\mathbf{A}$.

Sea $\mathbf{C}$ la matriz de cofactores de $\mathbf{A}$, es decir, cada elemento (i, j) de $\mathbf{C}$ corresponde al cofactor $C(\mathbf{A})_{ij} = (-1)^{i+j} \det(\mathbf{A}_{ij})$. Entonces, podemos definir a la *matriz adjunta* como la traspuesta de $\mathbf{C}$.

Esto es

$$\text{adj}\mathbf{A} = \begin{bmatrix} \mathbf{C}(\mathbf{A}_{11}) & \mathbf{C}(\mathbf{A}_{21}) & \cdots & \mathbf{C}(\mathbf{A}_{n1}) \\ \mathbf{C}(\mathbf{A}_{12}) & \mathbf{C}(\mathbf{A}_{22}) & \cdots & \mathbf{C}(\mathbf{A}_{n2}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}(\mathbf{A}_{1n}) & \mathbf{C}(\mathbf{A}_{2n}) & \cdots & \mathbf{C}(\mathbf{A}_{nn}) \end{bmatrix}$$

Ahora, como

$$\mathbf{A}(\text{adj}\mathbf{A}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} \mathbf{C}(\mathbf{A}_{11}) & \mathbf{C}(\mathbf{A}_{21}) & \cdots & \mathbf{C}(\mathbf{A}_{n1}) \\ \mathbf{C}(\mathbf{A}_{12}) & \mathbf{C}(\mathbf{A}_{22}) & \cdots & \mathbf{C}(\mathbf{A}_{n2}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}(\mathbf{A}_{1n}) & \mathbf{C}(\mathbf{A}_{2n}) & \cdots & \mathbf{C}(\mathbf{A}_{nn}) \end{bmatrix}$$

$$= \begin{bmatrix} \det(\mathbf{A}) & 0 & \cdots & 0 \\ 0 & \det(\mathbf{A}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \det(\mathbf{A}) \end{bmatrix} = \det(\mathbf{A})\mathbf{I}$$

Pues es fácil ver que al multiplicar el renglón i de $\mathbf{A}$ por la columna j de $\text{adj}\mathbf{A}$ se tiene la suma $\sum_{k=1}^{n} a_{ik}C(\mathbf{A})_{jk}$, por lo que sí $i = j$ nos queda la expansión de $\det(\mathbf{A})$ sobre el rengón i de $\mathbf{A}$, por otro lado, sí $i \neq j$ la suma es igual a 0. Ánalogamente para $(\text{adj}\mathbf{A})\mathbf{A} = \det(\mathbf{A})\mathbf{I}$.

Por consiguiente, sí $\det \neq 0$, entonces podemos calcular su *invesa* con

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})}\text{adj}\mathbf{A}.$$

Ya que

$$\mathbf{A}\Big(\frac{1}{\det(\mathbf{A})}\text{adj}\mathbf{A}\Big) = \frac{1}{\det(\mathbf{A})}[\mathbf{A}(\text{adj}\mathbf{A})] = \frac{1}{\det(\mathbf{A})}\det(\mathbf{A})\mathbf{I} = \mathbf{I}$$

Y como por definición tenemos que $\mathbf{AB} = \mathbf{I} \Rightarrow \mathbf{B} = \mathbf{A}^{-1}$. Entonces $\dfrac{1}{\det(\mathbf{A})}\text{adj}\mathbf{A} = \mathbf{A}^{-1}$.

En particular, calcular la matriz inversa de una matriz 2×2 es muy sencillo. Consideremos la matriz $\mathbf{A}$ de 2×2 y con $\det(\mathbf{A}) \neq 0$, entonces su matriz adjunta es

$$\text{adj}\mathbf{A} = \begin{bmatrix} \mathbf{C}(\mathbf{A}_{11}) & \mathbf{C}(\mathbf{A}_{21}) \\ \mathbf{C}(\mathbf{A}_{12}) & \mathbf{C}(\mathbf{A}_{22}) \end{bmatrix} = \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

por lo que se tiene

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})}\text{adj}(\mathbf{A}) = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}.$$

A continuación se muestra un ejemplo del cálculo de la inversa de una matriz.



$$\text{Sea } \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 5 & 8 & 9 \end{bmatrix}$$

1.Determinar si $\det(\mathbf{A}) \neq 0$.

$$\begin{vmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 5 & 8 & 9 \end{vmatrix}$$

Prev

Sig

Reset

Otro

Figura 2.25. Ejemplo del cálculo de la matriz inversa de una matriz $\mathbf{A}$. Se describe el proceso, y se puede observar cada paso usando los botónes *Prev*, *Sig* y *Reset* (para regresar al primer paso). O bien, ver el desarrollo con otra matriz presionado sobre el botón *otro*.

2.4.4 Cambio de base

Existen un número infinito de bases con las que podemos trabajar, ya que como vimos en el tema de *Espacios vectoriales, bases y coordenadas*, dentro de un espacio vectorial de dimensión n, cualesquiera n vectores *linealmente independientes* pueden formar una base.

Consideremos a los conjuntos de vectores $S_1 = \{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n\}$ y $S_2 = \{\widehat{\mathbf{e}}_1, \widehat{\mathbf{e}}_2, ..., \widehat{\mathbf{e}}_n\}$ como dos bases del espacio vectorial V n-dimensional.

Entonces podemos tomar a un vector cualquiera $\mathbf{u} \in V$, el cual puede ser descrito en términos de los vectores de la base S_1 como

$$\mathbf{u}_{S_1} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}; \quad \text{es decir, } \mathbf{u} = u_1 e_1 + u_2 e_2 + ... + u_n e_n$$

De manera similar, ese mismo vector puede ser expresado usando la base S_2 como

$$\mathbf{u}_{S_2} = \begin{bmatrix} \widehat{u}_1 \\ \widehat{u}_2 \\ \vdots \\ \widehat{u}_n \end{bmatrix}; \quad \text{es decir, } \mathbf{u} = \widehat{u}_1 \widehat{e}_1 + \widehat{u}_2 \widehat{e}_2 + ... + \widehat{u}_n \widehat{e}_n$$

Ahora, dependiendo de lo que se busque podemos construir la matriz de cambio de base. Llamaremos a S_1 como la base "original" y a S_2 como la base "nueva".

Definimos la *matriz de cambio de base* $\mathbf{P}$ como la matriz que transforma las coordenadas de $\mathbf{u}_{S_2}$ a coordenadas de la base original, esto es

$$\mathbf{P}\mathbf{u}_{S_2} = \mathbf{u}_{S_1}$$

Para ello, escribimos a cada vector $\widehat{\mathbf{e}}_i$ de la base nueva S_2 en términos de la base original S_1, es decir

$$\begin{aligned}\widehat{\mathbf{e}}_1 &= b_{11}\mathbf{e}_1 + b_{21}\mathbf{e}_2 + ... + b_{n1}\mathbf{e}_n \\ \widehat{\mathbf{e}}_2 &= b_{12}\mathbf{e}_1 + b_{22}\mathbf{e}_2 + ... + b_{n2}\mathbf{e}_n \\ &\vdots \\ \widehat{\mathbf{e}}_n &= b_{1n}\mathbf{e}_1 + b_{2n}\mathbf{e}_2 + ... + b_{nn}\mathbf{e}_n\end{aligned}$$

Por lo que si sustituímos en $\mathbf{u}_{S_2}$, tenemos

$$\begin{aligned}\mathbf{u} &= \widehat{u}_1(b_{11}\mathbf{e}_1 + b_{21}\mathbf{e}_2 + ... + b_{n1}\mathbf{e}_n) \\ &+ \widehat{u}_2(b_{12}\mathbf{e}_1 + b_{22}\mathbf{e}_2 + ... + b_{n2}\mathbf{e}_n) + \\ &\vdots \\ &+ \widehat{u}_n(b_{1n}\mathbf{e}_1 + b_{2n}\mathbf{e}_2 + ... + b_{nn}\mathbf{e}_n)\end{aligned}$$

Entonces

$$\mathbf{P} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Así

$$\mathbf{P}\mathbf{u}_{S_2} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} \begin{bmatrix} \widehat{u}_1 \\ \widehat{u}_2 \\ \vdots \\ \widehat{u}_n \end{bmatrix} = \mathbf{u}_{S_1}$$

Análogamente, definimos como *matriz de transición* $\mathbf{Q}$ como la matriz que transforma las coordenadas de $\mathbf{u}$ descrito con la base original S_1, a coordenadas de la nueva base, es decir,

$$\mathbf{Q}\mathbf{u}_{S_1} = \mathbf{u}_{S_2}$$

Escribiendo ahora a los vectores $\mathbf{e}_i$ como combinación lineal de la base nueva

$$\mathbf{e}_j = b_{1j}\widehat{\mathbf{e}}_1 + b_{2j}\widehat{\mathbf{e}}_2 + ... + b_{nj}\widehat{\mathbf{e}}_n$$

Podemos entonces obtener a $\mathbf{Q}$, siendo las columnas de la matriz iguales a las coordendas dadas por los vectores columna $\widehat{\mathbf{e}}_j$.

Nótese que $\mathbf{Q} = \mathbf{P}^{-1}$ y del mismo modo $\mathbf{P} = \mathbf{Q}^{-1}$, pues si multiplicamos $\mathbf{u}_{S_1} = \mathbf{P}\mathbf{u}_{S_2}$ por $\mathbf{P}^{-1}$ tenemos

$$\mathbf{P}^{-1}\mathbf{u}_{S_1} = \mathbf{P}^{-1}\mathbf{P}\mathbf{u}_{S_2} = \mathbf{I}\mathbf{u}_{S_2} = \mathbf{u}_{S_2}$$

2.5 Líneas y Planos

2.5.1 Líneas

Una *línea recta* puede ser definida por un punto inicial S y una dirección $\mathbf{d}$. Por consiguiente, si tomamos un punto P, y éste se encuentra sobre la línea, entonces el vector $\vec{SP}$ es paralelo a $\mathbf{d}$, y además, podemos asegurar que existe un escalar t tal que

$$\vec{SP} = t\mathbf{d}$$

Por lo que cualquier punto en la línea puede ser descrito con un $t \in \mathbb{R}$ único, lo cual suele ser muy útil. Denotemos entonces como $P(t)$ a la función paramétrica que nos devuelve el punto P sobre la línea dado un escalar t, la cual puede ser expresada como

$$\begin{gathered} \vec{SP}(t) = t\mathbf{d} \\ \Longleftrightarrow \\ P(t) - S = t\mathbf{d} \\ \Longleftrightarrow \\ P(t) = S + t\mathbf{d} \end{gathered}$$

Observemos en la **Figura 2.26** que si P está sobre la línea y se encuentra posicionado en la misma dirección que $\mathbf{d}$, entonces $t > 0$, de lo contrario, quiere decir que se encuentra en dirección opuesta a $\mathbf{d}$ y $t < 0$. De este modo, podemos obtener cualquier punto sobre una línea que se extiende hacia el infinito en ambas direcciones, al variar t entre $(-\infty, \infty)$.

Por otro lado, si se quiere describir un *rayo* que va desde el punto inicial S y se extiende hacia el infinito en dirección de $\mathbf{d}$, basta delimitar a t con un valor mayor o igual a 0. Se puede ver un ejemplo en la **Figura 2.26** como la línea azul.

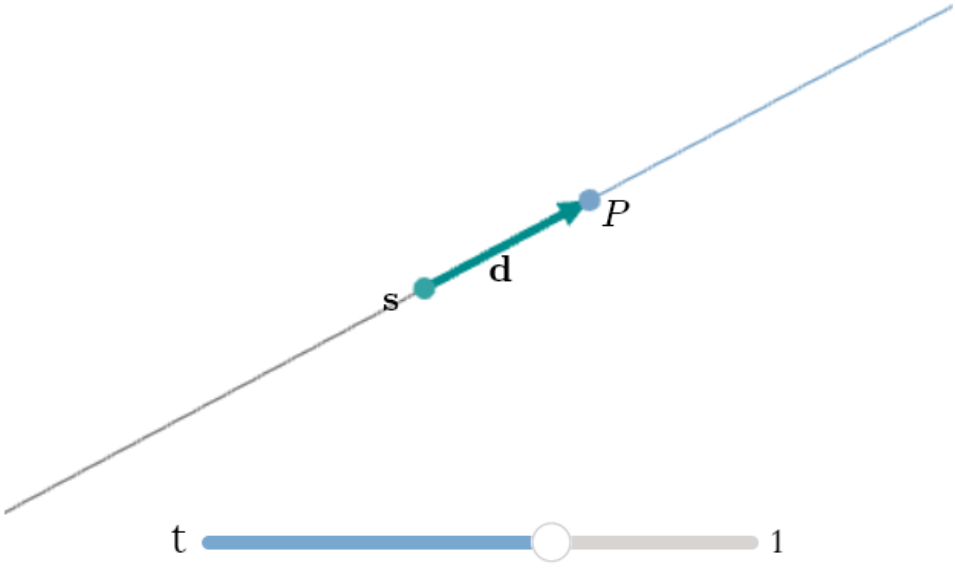


Figura 2.26. Visualización de la ecuación paramétrica $P(t) = S + t\mathbf{d}$.

Podemos definir también a la línea recta o *segmento de línea* que pasa entre dos puntos P_0 y P_1, al simplemente reemplazar a S por P_0 y a $\mathbf{d}$ por el vector $P_1 - P_0$, quedando nuestra ecuación como sigue

$$P(t) = (1 - t)P_0 + tP_1 \text{ con } t \in [0, 1].$$

Notemos que si tomamos la longitud de ambos lados de la ecuación $P(t) - S = t\mathbf{d}$, tenemos que $\|P(t) - S\| = \|t\mathbf{d}\|$, pero si normalizamos a nuestro vector de dirección, entonces nos queda como $\|P(t) - S\| = \|t\|$, simplificando la búsqueda de t, es decir, el punto $P(t)$ está situado a t unidades de S.

Vale la pena mencionar que las líneas en un espacio de *dimensión* 2 pueden ser representadas por un punto inicial S y su respectivo vector *normal* $\mathbf{n}$. Donde si un punto P se encuentra sobre la línea se cumple que

$$e(P) = \mathbf{n} \cdot (P - S) = 0$$

con $\mathbf{n} = (d_y, -d_x)$, pues $\mathbf{n} \cdot \mathbf{d} = d_y d_x - d_x d_y = 0$. (Ver **Figura 2.27**)

Esta función es conocida como *edge equation*, y ya que utliza el producto punto podemos ver que si $e(P) > 0$, entonces el ángulo más pequeño θ formado entre $\mathbf{n}$ y $(P - S)$ es menor que $\frac{\pi}{2}$, cayendo en el *lado o mitad positiva de la línea*, análogamente si $e(P) > 0$, entonces el ángulo es mayor que $\frac{\pi}{2}$, y cae en el *lado o mitad negativa de la línea*.

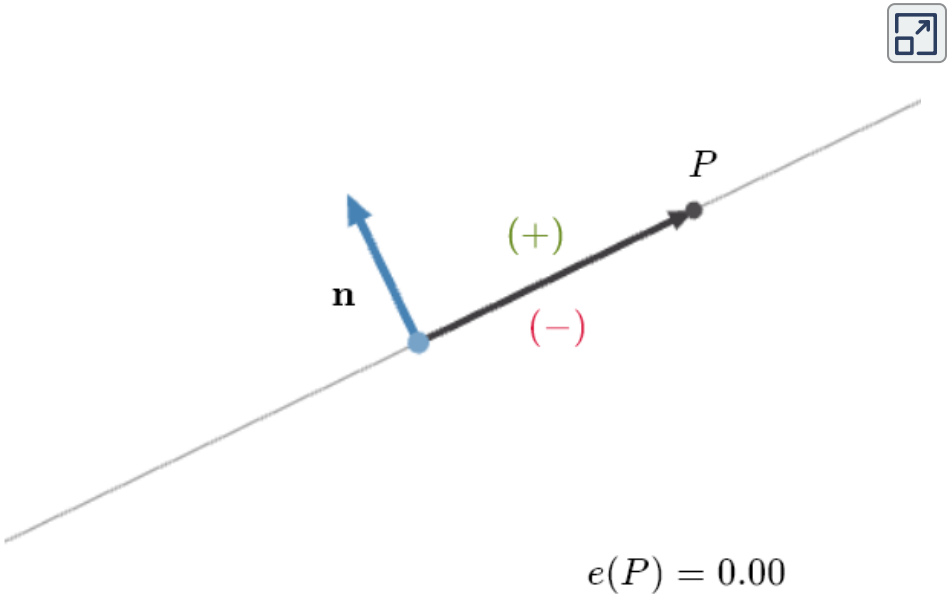


Figura 2.27. Visualización de la ecuación implícita $e(P) = \mathbf{n} \cdot (P - S) = 0$. Los signos de $+$ y $-$ muestran qué lado es la mitad positiva y negativa. Nótese que la línea gris equivale a los puntos donde $e(P) = 0$. En particular se anima al lector a mover el punto P y observar el resultado del cálculo de la ecuación $e(P)$.

Por último, si normalizamos a $\mathbf{n}$, vemos que $e(P) = \|(P - S)\| \cos\theta$, siendo ahora la distancia exacta (con signo) de la *proyección ortogonal* de P sobre la línea.

2.5.2 Planos

Un *plano* en tres o más dimensiones puede ser definido de manera similar que una línea bidimensional, esto es, con un punto inical S, y dos vectores de dirección no colineares entre sí **d1** y **d2**, los cuales estarán sobre el plano. Por lo que si un punto P se encuentra en el plano, entonces existen dos escalares $t_1, t_2 \in \mathbb{R}$ tales que

$$\overrightarrow{SP} = t_1\mathbf{d_1} + t_2\mathbf{d_2}$$

De modo que, podemos encontrar todos los puntos sobre el plano con la función paramétrica $P(t_1, t_2)$, donde

$$\begin{gathered}\overrightarrow{SP(t_1,t_2)} = t_1\mathbf{d_1} + t_2\mathbf{d_2} \\ \Longleftrightarrow \\ P(t_1,t_2) - S = t_1\mathbf{d_1} + t_2\mathbf{d_2} \\ \Longleftrightarrow \\ P(t_1,t_2) = S + t_1\mathbf{d_1} + t_2\mathbf{d_2}\end{gathered}$$

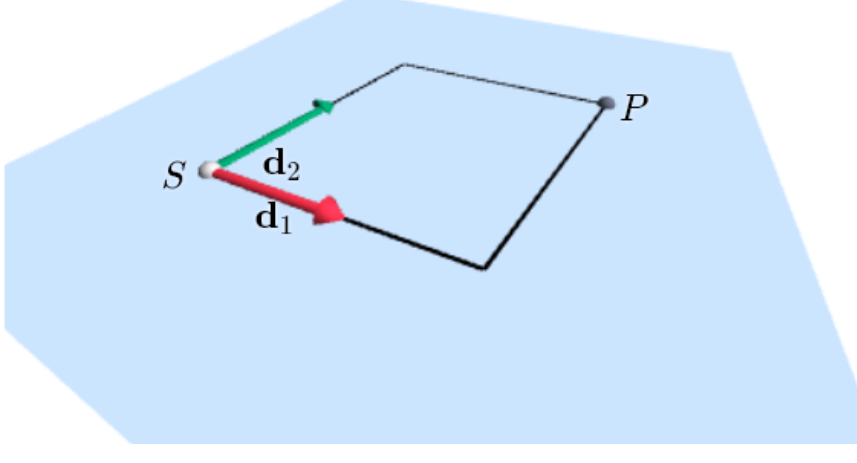


Figura 2.28. Punto P sobre un plano definido por los vectores de dirección $\mathbf{d}_1$ y $\mathbf{d}_2$, y su punto incial S. Nótese que es muy similar a la suma de dos vectores. El lector puede mover la posición de la cámara con el ratón o dedo, así como acercar o alejar la cámara con la rueda del ratón.

Notemos que sí alguno de los vectores de dirección es $\mathbf{0}$, o bien, $\mathbf{d1} = k\mathbf{d2}$ con $k \in \mathbb{R}$, la ecuación se reduce a la ecuación paramétrica de la línea.

Por otra parte, también podemos representar un plano con su punto incial S y su respectivo vector normal $\mathbf{n}$. Entonces, para todos los puntos P en el plano se cumple que

$$\mathbf{n} \cdot (P - S) = 0$$

Y ya que al igual que con las líneas, el plano divide el espacio de dominio en dos mitades, podemos crear del mismo modo la función de distancia $e(P) = \mathbf{n} \cdot (P - S)$.

Por lo tanto, si $e(P) = 0$ entonces P se encuentra sobre el plano. Ahora bien, si $e(P) > 0$ entonces P está del mismo lado que el punto $S + \mathbf{n}$ siendo la *mitad positiva del espacio*. Asimismo, si $e(P) < 0$ entonces P se encuentra del mismo lado que el punto $S - \mathbf{n}$, correspondiendo a la *mitad negativa del espacio*.

Otro aspecto útil en graficación es cómo obtenemos la *proyección ortogonal de un punto P sobre un plano*, el cual está definido por una normal $\mathbf{n}$. Para ello basta calcular el punto de proyección Q de la siguiente forma

$$\begin{gathered} Q = P - P_{\mathbf{n}}(P - S) \\ \iff \\ Q = P - P_{\mathbf{n}}\vec{SP} \end{gathered}$$

siendo S el punto incial. Este proceso se muestra en la **Figura 2.29**.

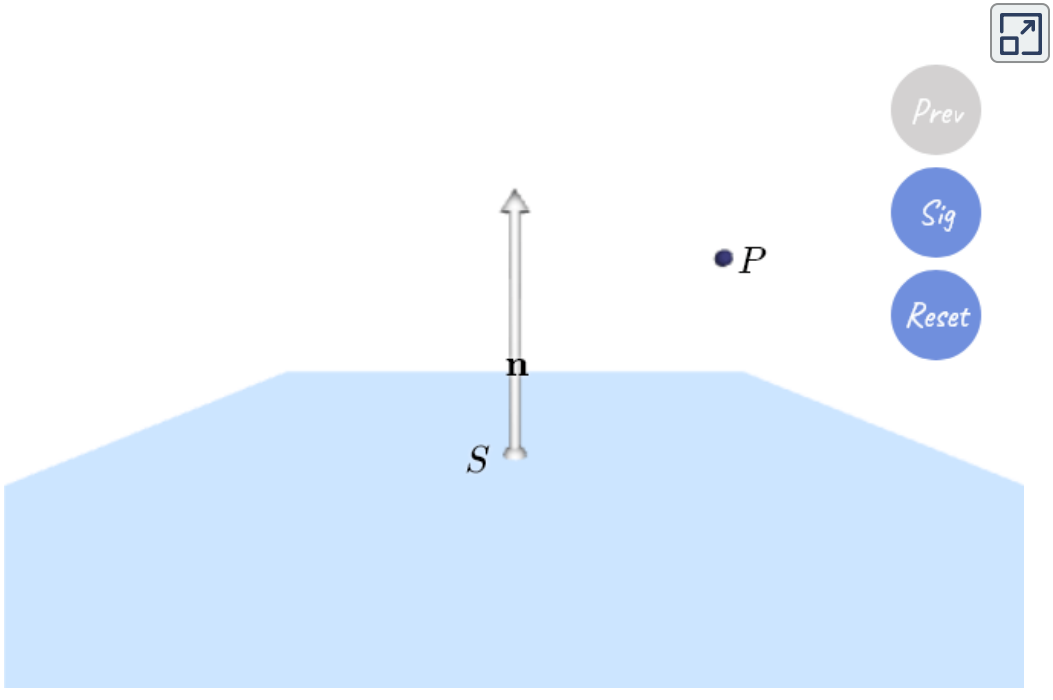


Figura 2.29. Proyección ortogonal de un punto P sobre un plano. El lector puede mover la cámara con el ratón o dedo, así como acercar o alejar la cámara con la rueda del ratón. Da clic en el botón *Sig* o *Prev* para ver el paso siguiente o previo.

2.6 Referencias bibliográficas

[38] Störm, Jacob, et al. ***Inmersive Linear Algebra***. 2ª Edición. 2011. http://immersivemath.com/

[39] Shirley, Peter, et al. ***Fundamentals of Computer Graphics.*** A K Press. 3ª Edición. 2009.

[40] Lengye, Eric. ***Mathematics for 3D Game Programming and Computer Graphics.*** Course Technology, a part of Cengage Learning. 3ª Edición. 2012.

[41] Seymour Lipschutz, et al. ***Linear Algebra.*** McGraw-Hill. 4ª Edición. 2009.

[42] Thomas Jr, George B. ***Cálculo de una variable.*** Addison-Wesley. 12va Edición. 2010.

[43] Grossman, Stanley I. ***Álgebra Lineal.*** McGraw-Hill. 6ª Edición. 2007.

[44] Grossman, Stanley I. ***Álgebra Lineal.*** Grupo Editorial Iberoamérica. 2ª Edición. 1987.

[45] Rose, David. 1 de Julio del 2009. ***Linear algebra for game developers ~ part 1***. Wolfire. http://blog.wolfire.com/2009/07/linear-algebra-for-game-developers-part-1/

[46] ***Line–line intersection***. 19 de Noviembre del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Line%E2%80%93line_intersection&oldid=989548298

[47] Page, John. 2011. ***Intersection of two straight lines***. Math Open Reference. https://mathopenref.com/coordintersection.html

[48] Belousov, Boris. 31 de Mayo del 2016. ***Change of basis vs linear transformation***. Boris Belousov. http://boris-belousov.net/2016/05/31/change-of-basis/

[49] 3Blue1Brown. 11 de Septiembre del 2016. ***Change of basis | Chapter 13, Essence of linear algebra***. [Archivo de Vídeo]. Youtube. https://youtu.be/P2LTAUO1TdA

CAPÍTULO III

Modelado geométrico

En graficación por computadora necesitamos una forma de representar o describir la geometría de una escena. El modelado geométrico estudia justo esto, cómo representar, crear y modificar dichas formas.

3.1 Vértices, aristas y caras

Las primitivas geométricas o de dibujo son las formas geométricas más simples (atómicas) con las que se puede dibujar. Las primitivas más usadas por la mayoría del *hardware* gráfico son: los *puntos* (vértices), las *líneas* (aristas) y los *triángulos* (caras).

Los puntos o vértices nos permiten representar localidades utilizando vectores de 2 o 3 dimensiones, un ejemplo, es el uso de estas primitivas para modelar sistemas de partículas, donde cada partícula es representada por un punto.



Figura 3.1. Ejemplo de un sistema de partículas usando puntos.

Por otra parte, una línea o arista, conecta dos vértices, de manera que podemos formar distintas figuras con un conjunto de líneas, al unir una serie de puntos, así como también modelar un sistema de partículas, o representar campos vectoriales en 2D y 3D, asociando cada vector con una línea.

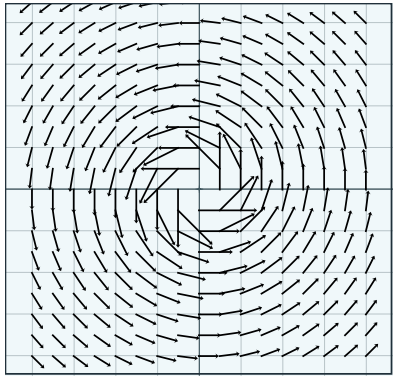


Figura 3.2. Visualización de un campo vectorial usando líneas.

Por último, tenemos a los triángulos, donde cada triángulo consta de tres vértices unidos por tres aristas, formando así una cara. Se opto por esta figura en especial ya que tienen la propiedad de ser *coplanarios*, puesto que tres puntos definen un plano, evitando así ambigüedad en cómo deben ser dibujados, lo cual puede llevar a inconsistencias con el sombreado. Además de ser es el polígono más simple y que cualquier polígono puede ser triangulado.

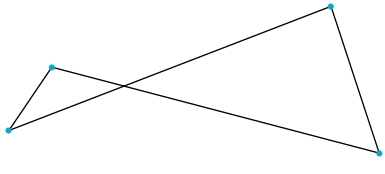


Figura 3.3. Ejemplo de figura (moño) formada con cuatro puntos no coplanarios.

3.2 Mallas poligonales

Intuitivamente, una malla poligonal, es una red de polígonos, los cuales están conectados entre sí por medio de vértices y aristas compartidas para formar así una superficie continua.

Las mallas poligonales surgen naturalmente de la necesidad de representar objetos más complejos, las más usadas en graficación son las mallas triangulares (*triangle mesh*) o cuadrangulares (*quad mesh*). Sin embargo, a la hora de renderizar, siempre se utilizan las primitivas de dibujo, por lo que los cuadriláteros terminarían siendo divididos en dos triángulos, convirtiéndose en una malla de triángulos. Por esta razón, cuando mencionemos una *malla* nos referiremos a una malla triangular.

Bunny. Modelo obtenido de *Stanford University Computer Graphics Laboratory*.

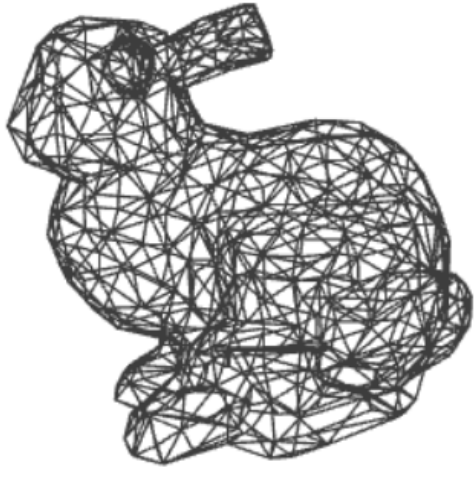


Figura 3.4. Ejemplo de modelo descrito con una malla triángular. Puedes mover la posición de la cámara con el ratón o dedo, así como alejarla o acercarla con la rueda del ratón.

Las mallas poligonales son una de las representaciones más usadas, y una de las razones principales es porque es sencillo convertir otro tipo de representación a una malla poligonal. Además de que dibujar triángulos es más fácil de optimizar que dibujar alguna otra figura más compleja. Como consecuencia las tarjetas de video han evolucionado en esta dirección para renderizar de forma más eficiente las mallas poligonales.

Por otro lado, una limitación que tiene al ser una representación discreta, es que las mallas o modelos representan las superficies curvas de manera aproximada, no obstante, podemos observar con el interactivo que con un buen

número de caras planas, podemos visualizar superficies curvas sin ningún problema.

Otra desventaja es que no puede compactarse, por lo que para representar un modelo muy detallado se necesitará una gran cantidad de datos, como podemos observar en la figura **Figura 3.5**. Además de que no es tan sencillo de modelar a comparación de otro tipo de representación como superficies implícitas o paramétricas.

Suzanne. Modelo obtenido de Blender.



Figura 3.5. Modelo Suzanne de Blender. Malla con 290 caras triangulares (izquierda) y 15,744 caras triangulares (derecha). El lector puede mover la cámara, así como alejarla o acercarla, esto último usando la rueda del ratón.

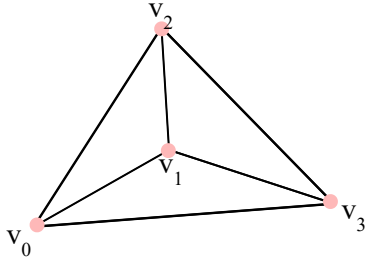
La descripción mínima de una malla es con un conjunto de triángulos y las posiciones de los vértices que los conforman. Aunado a esto, la mayoría de los programas suelen agregar más información en los vértices, aristas o caras, para poder realizar el sombreado (*shading*), mapeo de texturas y otro tipo de operaciones. La manera más común es guardar los atributos (posición, la normal, coordenadas de textura, colores) en cada vértice, donde cada atributo o parámetro cambia a lo largo de la superficie, al ser interpolados a lo largo de ésta.

Entonces, para poder representar una malla, necesitamos construir una estructura de datos que nos permita almacenar y organizar la descripción de la malla de manera eficiente. Si bien, existe una variedad de estructuras que podemos utilizar dependiendo del las operaciones que se van a efectuar sobre la malla, nosotros nos enfocaremos en dos de las más usadas.

Podemos representar a una malla triangular simple almacenando cada uno de los triángulos que la componen como triángulos independientes:

```
Triángulo {
  Vector3 coordenadas[3];
}
```

Por lo que su estructura de datos correspondería a una lista o un arreglo de triángulos, donde cada triángulo guarda las coordenadas de cada uno de sus vértices. A esta forma de almacenamiento se le conoce como *caras independientes*. Podemos ver un ejemplo con tan solo tres triángulos en la **Figura 3.6**, obsérvese que el vértice $\mathbf{v}_1$ es guardado 3 veces, y los otros 2 veces, sumando un total de 9 vértices (tres vértices por triángulo).



Caras independientes:

Triángulos	
0	$[v_0, v_1, v_2]$
1	$[v_1, v_3, v_2]$
2	$[v_0, v_3, v_1]$

Vértices compartidos:

Vértices	
0	v_0
1	v_1
2	v_2
3	v_3

Triángulos	
0	[0, 1, 2]
1	[1, 3, 2]
2	[0, 3, 1]

Figura 3.6. Malla triangular representada con caras independientes (Izquierda) y vértices compartidos (Derecha)

Otra forma de representar una malla es separando la información, almacenando a cada vértice como una estructura independiente y a los triángulos como otra que contenga los índices que hacen referencia a los vértices que lo conforman:

```
Triángulo {
  int índices[3];
}

Vértice {
  Vector3 coordenadas; // Aquí podemos agregar más atributos
}
```

entonces la estructura de datos correspondería a una lista o un arreglo de vértices y otra lista o arreglo de triángulos, de esta manera toda la información

de cada vértice compartido se almacena una sola vez y cada triángulo se representa con tres índices de la lista de vértices. A este tipo de estructura se le conoce como *indexed triangle mesh* o de *vértices compartidos*.

Como podemos notar, esta estructura nos permite ahorrar más espacio de almacenamiento, así como llevar un mejor control de los atributos del vértice. Por otra parte, un problema que tiene esta implementación es que no permite discontinuidad de color o alguna otra propiedad, ya que dos polígonos que comparten el mismo vértice, terminan utilizando la misma información, produciendo un efecto de suavizado en el color de la superficie. Un ejemplo sencillo es el que se muestra en la **Figura 3.7**.



Figura 3.7. (Caja azul) Normales compartidas, (Caja Roja) Normales independientes. Puedes mover la posición de la cámara con el ratón o dedo.

En donde podemos observar dos cajas. La caja azul utiliza esta misma implementación, provocando un efecto de suavizado en su color ya que los vértices de las esquinas comparten la misma normal. Y con la caja roja se soluciona este problema al crear una discontinuidad geométrica entre estos vértices, se separan las caras duplicando los vértices de las esquinas, pudiendo así calcular las normales correctamente y producir una buena iluminación.

Otra desventaja es que toma un poco más de tiempo en acceder a cada vértice, ya que primero se necesita ver el índice y luego con éste obtenerlo, a diferencia de la primera estructura, donde se puede acceder directamente.

3.2.1 Tira y abanico de triángulos

La forma de representar una malla de manera más compacta es usando las primitivas de *tira de triángulos* o *triangle strip* y abanico de triángulos o *triangle fan*.

El primero, consta de una secuencia de triángulos, en donde con los primeros tres vértices se construye el primer triángulo, y cada nuevo vértice define un nuevo triángulo tomando los dos últimos vértices del triángulo anterior. Para ser más específicos, dada la lista de vértices $\{\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_n\}$, el triángulo i está representado por los vértices $\{\mathbf{v}_i, \mathbf{v}_{i+1}, \mathbf{v}_{i+2}\}$ (Ver **Figura 3.8**).

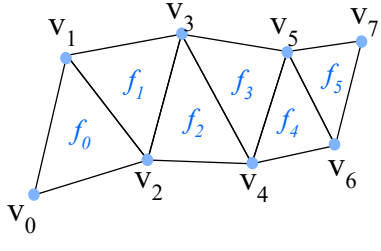


Figura 3.8. Tira de triángulos

Mientras que para la segunda primitiva, su primer vértice corresponde al vértice común que compartirán todos los triángulos, donde el triángulo i está representado por los vértices $\{\mathbf{v}_0, \mathbf{v}_{i+1}, \mathbf{v}_{i+2}\}$ (Ver **Figura 3.9**).

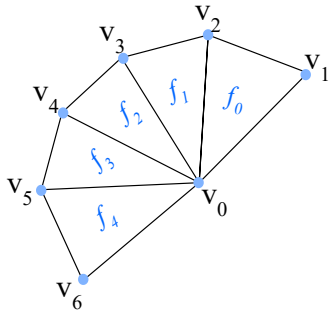
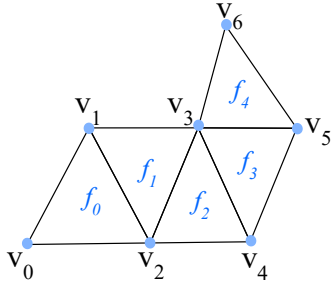


Figura 3.9. Abanico de triángulos

De tal forma que con n vértices se obtienen $n-2$ triángulos, usando cualquiera de las dos primitivas. Por otro lado, si el modelo esta formado por varios strips, éstos suelen unirse por medio de triángulos degenerados (triángulos con área igual a cero) como se muestra en la **Figura 3.10**.



Lista de triángulos = { (v_0, v_1, v_2),
(v_1, v_2, v_3),
(v_2, v_3, v_4),
(v_3, v_4, v_3),
(v_4, v_3, v_5),
(v_3, v_5, v_6) }

Figura 3.10. Representación de malla uniendo dos tiras de triángulos especificando al triángulo degenerado $(\mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_3)$en la lista de vértices.

3.2.2 Orientación

Otro punto importante a considerar es la orientación de las caras, para así distinguir el lado exterior o frontal del lado interior o trasero.

En particular, para un solo triángulo podemos definir su orientación de acuerdo al orden en que los vértices están listados, esto puede ser en sentido de las manecillas del reloj y en sentido contrario a las manecillas, como podemos observar en la **Figura 3.11**. Nótese que las caras f_1 y f_2 tienen la misma orientación sí y solo sí la arista compartida aparece descrita en cada cara con sus vértices en orden contrario. Por lo que la cara que se estaría viendo sería la frontal, y estaría definida en un cierto sentido, el cual suele ser en contra de las

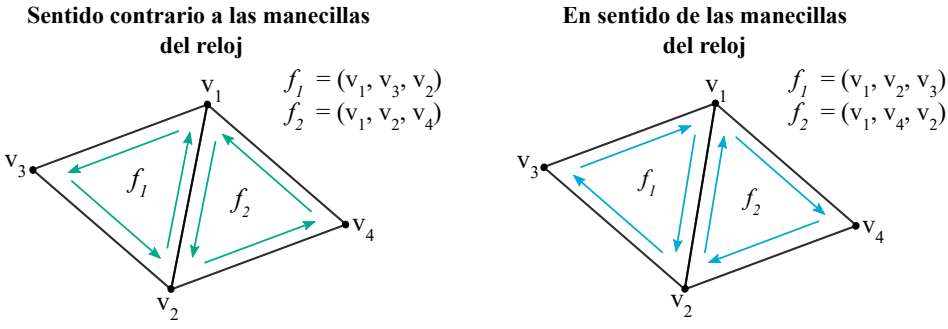


Figura 3.11. Orientación de las caras.

manecillas. A esto se lo conoce como *winding direction* y los triángulos usan la regla de la mano derecha.

Se dice que una malla está *orientada consistentemente* si todos los triángulos que la componen tienen la misma orientación, coincidiendo en qué lado es el frontal, y esto se cumple solamente sí cada par de triángulos adyacentes están orientados consistentemente. No obstante existen superficies no orientables, como la banda de Möbius.

Esta propiedad nos permite calcular las normales de las caras de manera correcta. Además de otras aplicaciones, como en el renderizado de objetos transparentes o para descartar aquellas caras de la malla que no sean visibles, ya que si una cara está definida en sentido contrario a las caras frontales, sabemos que se trata de una cara interior o trasera, la cual no es visible. Por ejemplo, si imaginamos un cubo sólido, sin importar su dirección, solo seremos capaces de ver a lo más 3 de sus caras.

3.3 Curvas de Bézier

Las curvas de Bézier son una de las representaciones más usadas en GC para trazar "curvas libres" (*free-form curves*). El algoritmo fue desarrollado de manera independiente por *Paul de Casteljau* y *Pierre Bézier* para ser usado dentro de la industria automotríz. Son llamadas curvas de Bézier debido a que las políticas de privacidad de la empresa en la que trabajaba de Casteljau le impidieron publicar su trabajo antes.

Una curva de Bézier es una curva polinomial que se aproxima a un conjunto de puntos, llamados *puntos de control*, una forma de representarlas es la siguiente.

Dados $n+1$ puntos $P_0, P_1, ..., P_n$ la curva de Bézier es definida por la ecuación paramétrica:

$$P(t) = \sum_{i=0}^{n} B_i^n(t) P_i \quad t \in [0,1]$$

donde P_i son los puntos de control y los coeficientes B_i^n son los polinomios de *Bernstein*.

Un polinomio de Bernstein de grado n se define como

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad i = 0, ..., n$$

donde $\binom{n}{i}$ es el coeficiente binomial, y se define como $\binom{n}{i} = \frac{n!}{(n-i)!i!}$, sin embargo, al tener factoriales se opta por utilizar el patrón de números producido por el triángulo de Pascal (**Figura 3.12**) y se calcula de manera recursiva.

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
  1   7  21  35  35  21   7   1
```

Figura 3.12. Triángulo de Pascal hasta n = 7.

Las propiedades de los polinomios de Bernstein nos dicen mucho sobre el comportamiento o forma de las curvas de Bézier, analizaremos algunas de ellas más adelante.

Se dice que una curva es de grado n si es controlada por $n+1$ puntos de control, por lo que mientras más puntos de control tenga, más flexibilidad tendrá la curva. Sin embargo, el uso de conjuntos grandes de puntos puede resultar muy costoso, ya que el grado de los polinomios aumentaría. Una solución sería uniendo curvas de grados menores.

Empecemos con dos puntos, siendo una curva de Bézier de grado 1. Sean P_0 y P_1 dos puntos de control, la curva es un segmento de línea descrito por la interpolación lineal del punto P_0 al punto P_1

$$\begin{aligned} P(t) =& B_0^1(t)P_0 + B_1^1(t)P_1 \\ =& \binom{1}{0} t^0 (1-t)^1 P_0 + \binom{1}{1} t^1 (1-t)^0 P_1 \\ =& (1-t)P_0 + tP_1 \quad t \in [0,1] \end{aligned}$$

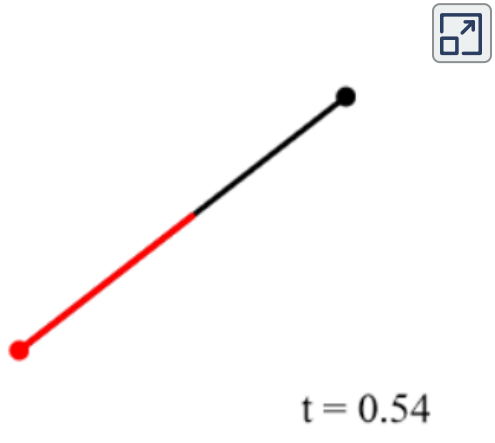


Figura 3.13. Curva de Bézier de grado 1: Interpolación lineal de dos puntos dado el parámetro t, con $t \in [0,1]$.

Siendo más específicos, si tomamos dos puntos de la forma $P_0 = (x_0, y_0)$ y $P_1 = (x_1, y_1)$, entonces las coordenadas del punto $P(t) = (x, y)$ en la curva están dadas por

$$x' = (1-t)x_0 + tx_1 \quad \text{y} \quad y' = (1-t)y_0 + ty_1$$

Análogamente para puntos tridimensionales, se realiza una interpolación lineal sobre cada una de sus coordenadas.

Dados los puntos de control P_0, P_1 y P_2, se define la curva de Bézier de grado 2 o cuadrática como

$$P(t) = B_0^2(t)P_0 + B_1^2(t)P_1 + B_2^2(t)P_2$$

$$
\begin{aligned}
&= \binom{2}{0} t^0 (1-t)^2 P_0 + \binom{2}{1} t^1 (1-t)^1 P_1 + \binom{2}{2} t^2 (1-t)^0 P_2 \\
&= (1-t)^2 P_0 + 2t(1-t) P_1 + t^2 P_2 \quad t \in [0,1]
\end{aligned}
$$

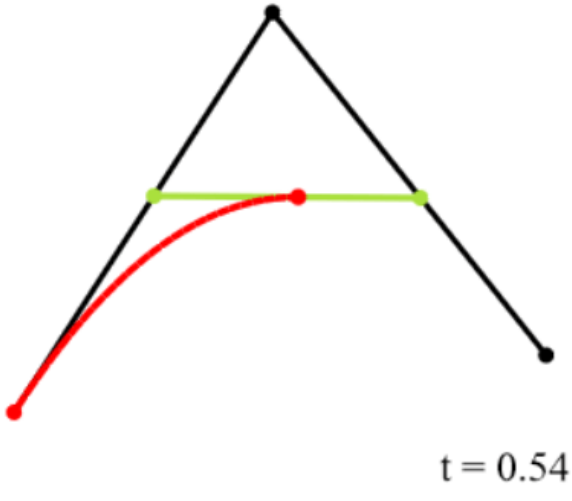


Figura 3.14. Algoritmo de Casteljau. Construcción de una curva de Bézier de grado 2, con $t \in [0,1]$.

esta expresión puede reescribirse de forma matricial como

$$
\begin{aligned}
P(t) &= \begin{bmatrix} (1-t)^2 & t(1-t) & t^2 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & t & t^2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix}
\end{aligned}
$$

donde se multiplica la base $\{1, t, t^2\}$ con los coeficientes obtenidos por los polinomios (desarrollados) de Bernstein, lo cual suele ser util para simplificar los cálculos en la implementación.

Dados los puntos de control P_0, P_1, P_2 y P_3, se define la curva de Bézier de grado 3 o cúbica como

$$
\begin{aligned}
P(t) =& B_0^3(t)P_0 + B_1^3(t)P_1 + B_2^3(t)P_2 + B_3^3(t)P_3 \\
=& \binom{2}{0} t^0 (1-t)^3 P_0 + \binom{3}{1} t^1 (1-t)^2 P_1 + \binom{3}{2} t^2 (1-t)^1 P_2 + \binom{3}{3} t^3 (1-t)^0 P_3 \\
=& (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \quad t \in [0,1]
\end{aligned}
$$

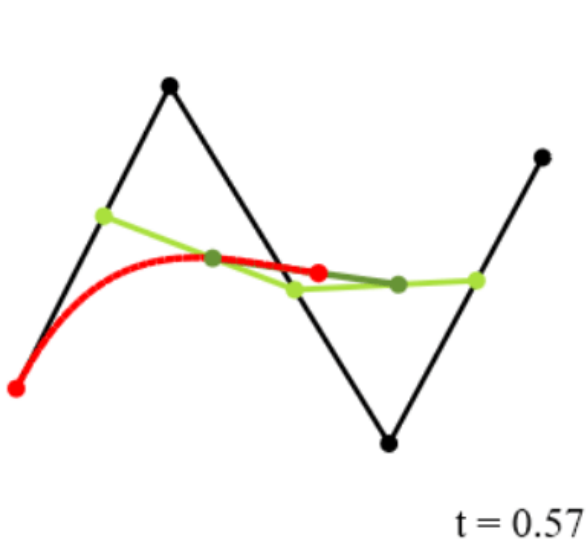


Figura 3.15. Algoritmo de Casteljau. Construcción de un curva de Bézier de grado 3, con $t \in [0,1]$.

y puede reescribirse de forma matricial como

$$
\begin{aligned}
P(t) &= \begin{bmatrix} (1-t)^3 & t(1-t)^2 & t^2(1-t) & t^3 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \\
&= \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & -3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}
\end{aligned}
$$

Como hemos podido observar de los interactivos por cada punto adicional, se agrega además un paso de interpolación.

El algoritmo descrito por los interactivos es el *algoritmo de Casteljau* y es una forma de trazar (o aproximar) las curvas de Bézier. Comienza entonces uniendo los puntos de control adyacentes para después encontrar el valor en t de cada segmento mediante la interpolación de sus extremos, dejando un conjunto de $n-1$ puntos. De nuevo, estos puntos son unidos con líneas y son interpolados en t, obteniendo un nuevo grupo de $n-2$ puntos. Y esto continua hasta terminar con un solo punto, el cual es el punto final sobre la curva.

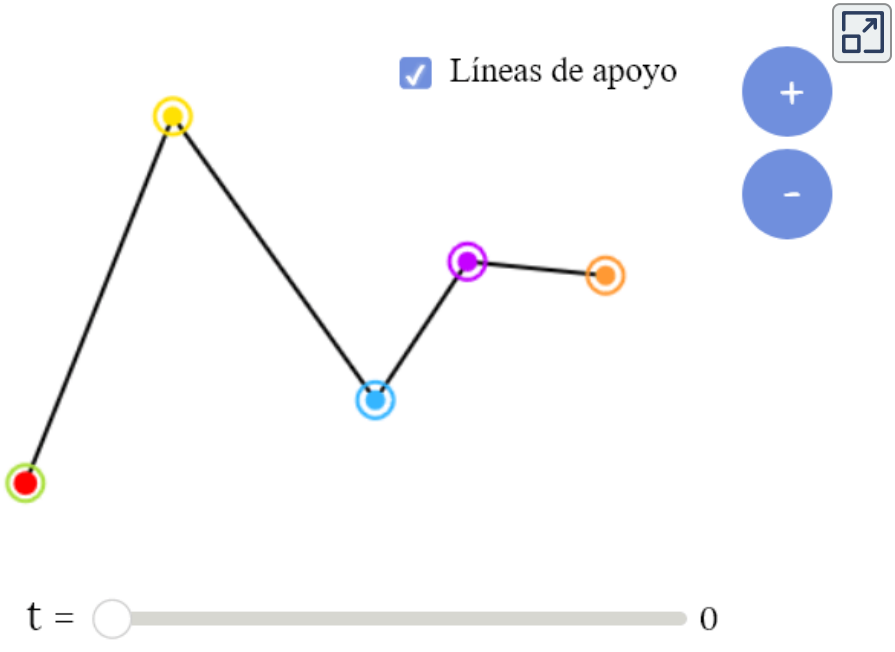


Figura 3.16. Curva de Bézier de grado 4. Se invita al lector a mover los puntos de control arrastrándolos, así como modificar el valor de t. Además de poder agregar puntos de control arbitrarios con el botón + y eliminar puntos de control con el botón − aumentando o disminuyendo el grado de la curva.

En la **Figura 3.17** se muestran los polinomios de Bernstein de grado 1, 2 y 3, los cuales también son llamados como *blending functions* o funciones mezcladoras, ya que especifican los valores de peso o influencia que tendrán sobre sus respectivos puntos de control. Pues como hemos podido notar en las fórmulas, cada uno de los puntos de control se corresponde con un polinomio de Bernstein. De modo que la influencia o pesos asociados a los puntos de control son mezclados para obtener los puntos finales en la curva.

Observando las gráficas nos podemos dar una idea de la influencia que tiene cada punto de control sobre cada punto de la curva.

Por ejemplo, para $n = 1$ (la interpolación lineal) **Figura 3.17** (a) se muestran las curvas $(1 - t)$ y t, vemos que $(1 - t)$ empieza en 1 y termina en 0, en cambio notamos que t empieza en 0 y termina en 1, esto quiere decir que cuando $t = 0$, entonces $t = P_0$, y cuando t incrementa, el peso o influencia de P_0 disminuye, mientras que el peso P_1 incrementa a medida que t se acerque a 1. Finalmente cuando $t = 1$ alcanza a P_1. Este comportamiento lo podemos apreciar en los interactivos.

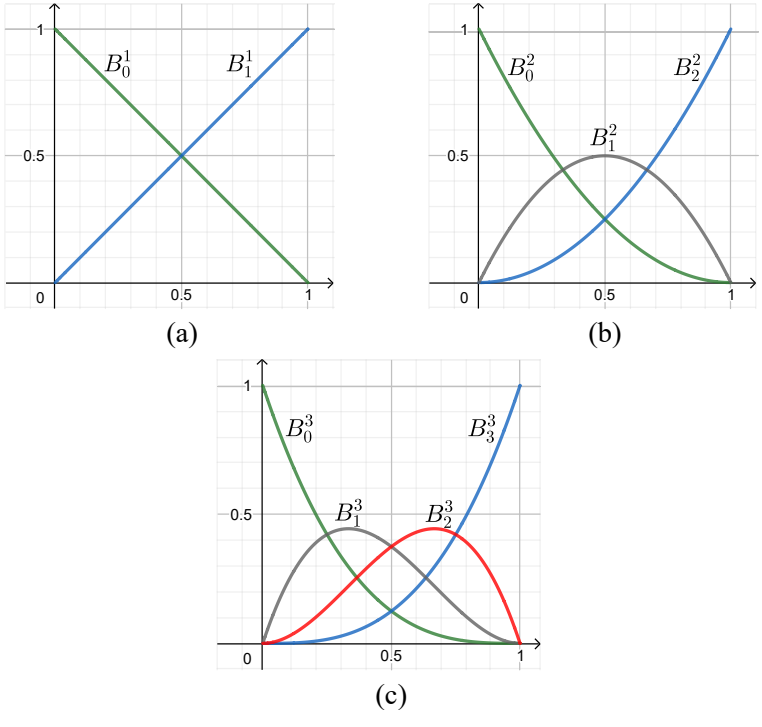


Figura 3.17. Polinomios de Bernstein de grado 1 (a), 2 (b) y 3 (c). Siendo las funciones de mezcladoras utilizadas para la construcción de curvas de Bézier. Graficado con Geogebra Copyright (C) International GeoGebra Institute, 2013.

En general esto se mantiene para todas las curvas de Bézier, es decir, siempre se cumple que $B_0^n(0) = 1$ y $B_n^n(n) = 1$, por lo que la curva solo interpola (pasa por) el primer y el último punto de control, ya que P_0 y P_n son los únicos puntos donde algún polinomio tiene un valor igual a 1. Además, la curva es tangente al vector $P_1 - P_0$ cuando $t = 0$ y al vector $P_n - P_{n-1}$ cuando $t = 1$.

Mientras que la forma de la curva es aproximada o influenciada por los otros puntos, por ejemplo para $n = 2$, **Figura 3.17** (b), el polinomio $(2t(1-t))$ empieza en 0, después alcanza su punto máximo en 0.5 y vuelve a terminar en 0, dándole así un cierto peso a P_1 cuando $0 < t < 1$. Análogamente con los polinomios de grado 3, **Figura 3.17** (c).

La *simetría* es una propiedad que podemos notar fácilmente e implica que podemos revertir el orden de los puntos de control y seguir obteniendo la misma curva. Esto, también se aprecia en las gráficas de la **Figura 3.17** al rededor de $t = 0.5$.

Una propiedad muy importante es la *partición de unidad*, esto es que los polinomios de Bernstein suman uno para cualquier valor t, lo cual podemos apreciar mediante la expansión binomial

$$\sum_{i=0}^{n} B_i^n(t) = [(1-t)+t]^n = 1$$

entonces cualquier punto en la curva (o la curva en sí) es una combinación convexa afín de los puntos de control. Por tanto, todos los puntos estarán dentro del polígono convexo formado por los puntos de control.

Y siendo una combinación afín, tiene una *invariancia afín*, esto es que podemos transformar una curva de Bézier α al aplicar una transformación afín T, es decir, una transformación lineal con una traslación (se verán más a detalle en el Capítulo 4), sobre cada uno de sus puntos, dicho de otro modo, la curva de Bézier asociada a los puntos de control $T(P_0), T(P_1), ..., T(P_n)$ es precisamente $T(\alpha)$. Lo cual es fundamental en graficación para la manipulación de puntos, permitiéndonos también ahorrar memoria al únicamente almacenar los puntos de control.

Además, podemos ver que cuando se mueve un punto de control, toda la curva es afectada, se dice entonces que los puntos tienen un *control global*. Lo cual no suele ser una característica muy deseada para diseñar curvas, pues se prefiere que cuando se edite algún punto de control varie sólo en cercanías del punto modificado, sin alterar la forma del resto de la curva. A esto se le conoce como *control local*, lo cual puede ser solucionado con las *splines* (funciones polinómicas a trozos).

3.3.1 Parches de Bezier

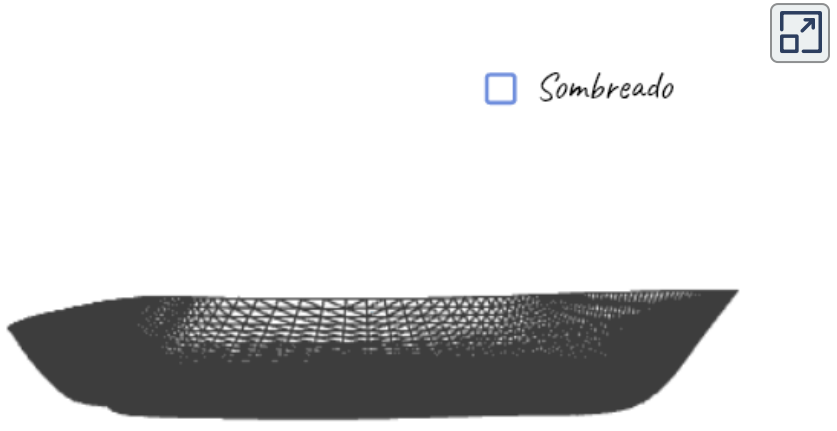


Figura 3.18. Canoa. Ejemplo de un modelo que se puede hacer con parches de Bézier. Puedes mover la posición de la cámara con el ratón o dedo, así como acercarte o alejarte con la rueda del ratón.

Los parches o superficies de Bézier son una extensión natural de las curvas de Bézier. Una analogía es un triángulo o un polígono, el cual es una extensión de una línea pasando de una a dos dimensiones. Ahora, en vez de interpolar una secuencia de puntos, podemos pensarlo como una manera de interpolar una secuencia de curvas de Bézier, necesitando ahora dos parámetros para definir la superficie.

Una superficie o parche de Bézier se define como el producto cartesiano o tensorial

$$P(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(u) B_j^n(v) P_{ij} \quad (u,v) \in [0,1]^2$$

donde B_i^m y B_j^n son los polinomios de Bernstein de grado m y n respectivamente, los cuales actúan como funciones mezcladoras sobre los puntos de control P_{ij}. Notemos que el domino de la superficie es rectangular. Por esta razón es que suelen ser llamados como parches.

Si mantenemos el parámetro u constante y recorremos la superficie con v de 0 a 1, se traza una curva de Bézier que se encuentra en la superficie, y lo mismo ocurre con el parámetro v.

Siendo más específicos, un parche de Bézier de grado $(m \times n)$ está definido por $(m+1) \times (n+1)$ puntos de control como

$$P(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \binom{m}{i} \binom{n}{j} u^i (1-u)^{m-i} v^j (1-v)^{n-j} P_{ij} \quad (u,v) \in [0,1]^2$$

Empecemos de nuevo con el más sencillo, extendiendo la interpolación lineal a una interpolación *bilineal*. Es decir, en lugar de interpolar dos puntos, se interpolan cuatro y en lugar de utilizar al parámetro t usaremos dos parámetros u y v. Sean P_{00}, P_{01}, P_{10} y P_{11} los puntos de control, entonces u interpola a P_{00} con P_{01} y a P_{10} con P_{11} como sigue

$$Q_0 = (1-u)P_{00} + uP_{01} \quad Q_1 = (1-u)P_{10} + u^P_{11}$$

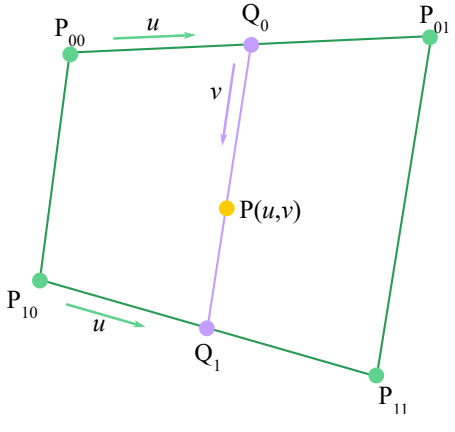


Figura 3.19. Parche de Bézier de grado (1×1). Interpolación bilineal usando cuatro puntos.

Finalmente los puntos obtenidos Q_0 y Q_1 son interpolados por el parámetro v, teniendo

$$\begin{aligned} P(u,v) &= (1-v)Q_0 + vQ_1 \\ &= (1-u)(1-v)P_{00} + u(1-v)P_{01} + (1-u)vP_{10} + uvP_{11} \end{aligned}$$

con los valores de $(u,v) \in [0,1]^2$.

Y así como con las curvas de Bézier, donde para obtener los puntos sobre la curva se realizan repetidas interpolaciones lineales. Ahora, para obtener las superficies de Bézier se realizan repetidas interpolaciones bilineales.

Supongamos que queremos obtener los puntos de un parche de grado (2×2) o bicuadrático definido por un arreglo rectangular de 3×3 puntos de control P_{ij}. Entonces, para encontrar un punto sobre la superficie dados u y v, primero se interpolan bilinealmente los puntos de control cercanos para así obtener cuatro puntos intermedios P_{ij}^1, ver **Figura 3.20**. Formando ahora un parche de Bézier de 2×2 puntos. Por último esos cuatro puntos son interpolados bilinealmente para encontrar el punto sobre la superficie P_{00}^2.

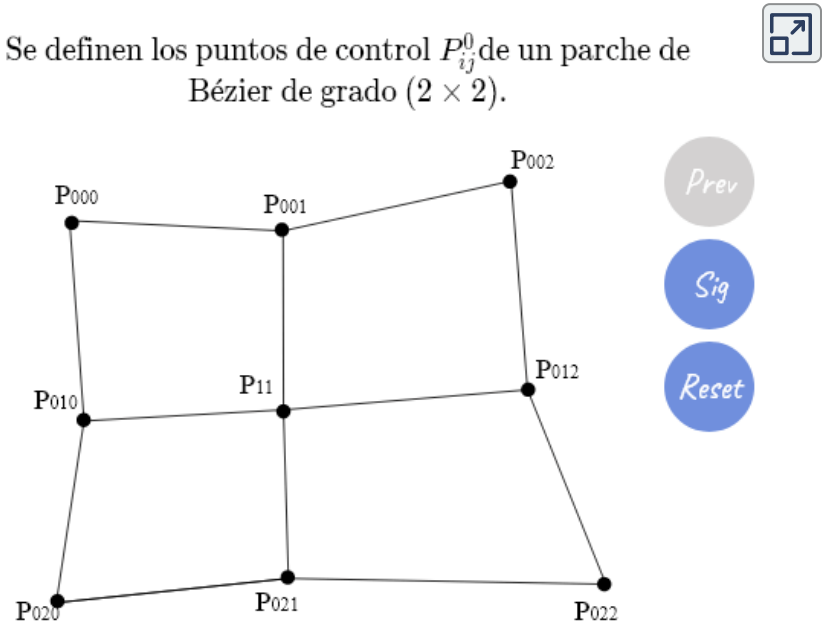


Figura 3.20. Algoritmo de Casteljau. Proceso para obtener un punto $P(u, v) = P_{00}^2$ sobre un parche de Bézier grado (2×2) definido con nueve puntos de control P_{ij}^0.

Este proceso recursivo es una extensión del *algoritmo de Casteljau* y se aplica de manera análoga para parches de Bézier de grado $m \times n$ con $m = n$, teniendo puntos de control P_{ij}, con $i, j \in \{0, .., n\}$. Este caso suele ser el más común y simplifica un poco la implementación.

Sin embargo, también podemos tener el caso en el que tengamos distintos grados para m y n, en particular, con puntos de control P_{ij} con $i \in \{0, ..., m\}$ y $j \in \{0, ..., n\}$. Para este tipo de superficies el algoritmo necesita tener una variación. De la **Figura 3.21** podemos ver que con el algoritmo de Casteljau no se puede llegar a encontrar al punto sobre la superficie, ya que después de haberlo realizado $k = \min(m, n)$ veces, los puntos intermedios P_{ij}^k terminan formando una polígono de control de forma curva. Por lo que ahora se procede a encontrar al punto en la superficie con el algoritmo univariable de Casteljau, es decir, para curvas.

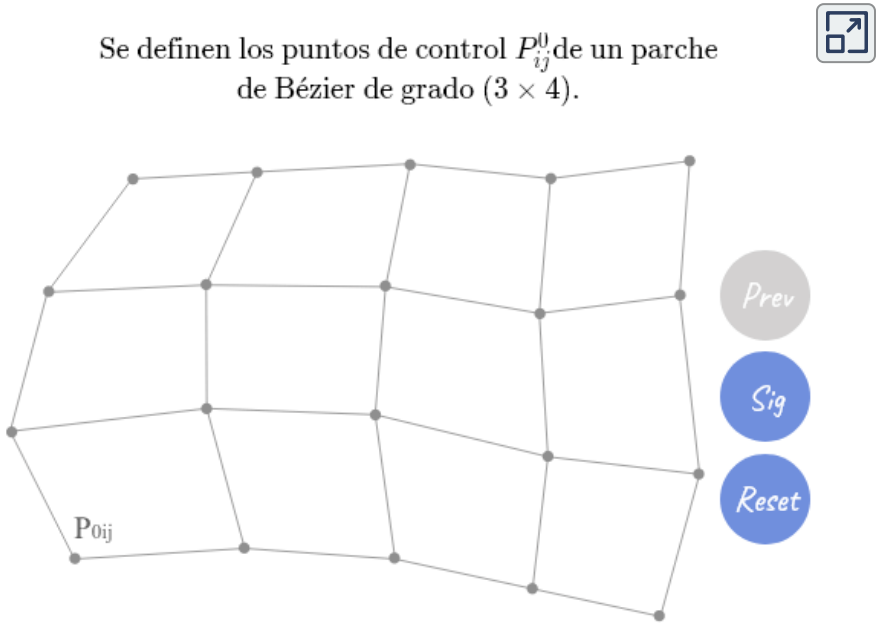


Figura 3.21. Algoritmo de Casteljau. Proceso para obtener al punto $P(u, v) = P_{00}^4$ sobre un parche de Bézier de grado (3×4).

Por último, ya que los parches son una extensión natural de las curvas de Bézier, éstos heredan sus mismas propiedades, enlistadas a continuación

i. El parche interpola las esquinas de sus puntos de control, esto se puede ver fácilmente con $(u, v) = (0, 0)$, $(u, v) = (0, 1)$, $(u, v) = (1, 0)$ y $(u, v) = (1, 1)$.

ii. La superficie está siempre dentro del polígono convexo formado por los puntos de control.

ii. Es invariante bajo transformaciones afines, es decir, podemos transformar la superficie aplicando la función de transformación sobre sus puntos de control.

iii. La superficie es tangente a los vectores formados por los puntos de las esquinas y los puntos adyacentes.

iv. El parche puede ser computado y/o subdividido recursivamente con el algoritmo de Casteljau.

Por otra parte, para construir objetos más complicados varios parches de Bézier pueden ser unidos o "cosidos" para formarlos.

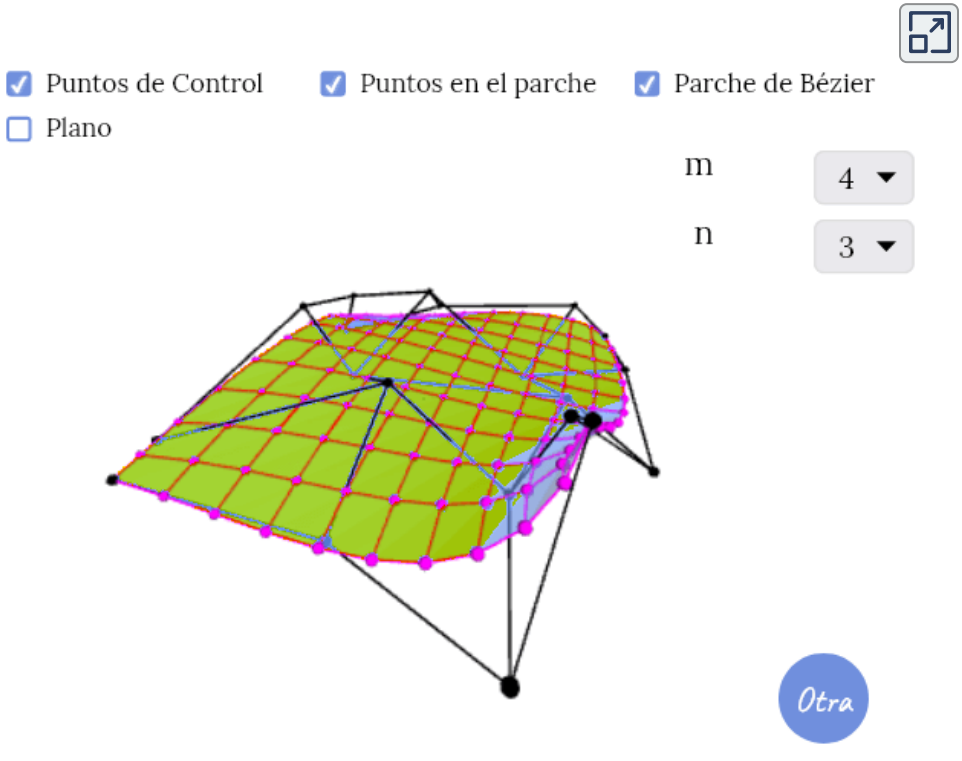


Figura 3.22. Parche de Bézier. Se invita al lector a cambiar de superficie dando clic en el botón *Otra*, o bien modificando las dimensiones de m y n. Modifica la visualización de la superficie dando clic sobre las opciones: *Puntos de control*, *Puntos en el parche* y/o *Parche de Bézier*. Puedes mover la posición de la cámara con el ratón o dedo, así como acercar y alejarte con la rueda del ratón.

3.4 Referencias bibliográficas

[50] Akenine-Möller, Tomas, et al. ***Real-Time Rendering.*** A K Peters/CRC Press. 4ª Edición. 2018.

[51] Shirley, Peter, et al. ***Fundamentals of Computer Graphics.*** A K Press. 3ª Edición. 2009.

[52] Gortle, Steven J. ***Foundations of 3D Computer Graphics.*** MIT press. 1ª Edición. 2011.

[53] Ribelles, José, et al. ***Informática Gráfica.*** Publicacions de la Universitat Jaume I. 2ª Edición. 2019.

[54] Lengye, Eric. ***Mathematics for 3D Game Programming and Computer Graphics.*** Course Technology, a part of Cengage Learning. 3ª Edición. 2012.

[55] Ganovelli, Fabio, et al. ***Introduction to Computer Graphics a Practical Learning Approach.*** CRC Press. 2015.

[56] Dunn, Fletcher, et al. ***3D Math Primer for Graphics and Game Development.*** CRC Press. 2ª Edición. 2011.

[57] Ferguson, R. Stuart. ***Practical Algorithms for 3D Computer Graphics.*** A K Peters/CRC Press. 2ª Edición. 2013.

[58] Vince, John. ***Mathematics for Computer Graphics.*** Springer-Verlag. 5ª Edición. 2017.

[59] Farin, Gerald. ***Curves and Surfaces for CAGD. A Practical Guide.*** Morgan Kaufmann Publishers. 5ª Edición. 2013.

[60] Telea, Alexandru C. ***Data Visualization Principles and Practice.*** A K Peters/CRC Press. 2ª Edición. 2014.

[61] ***Geometric primitive***. 17 de Septiembre del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Geometric_primitive&oldid=978917463

[62] Fernández, Marco. Coma, Inmaculada. 2007. **TEMA 1: MODELOS DE REPRESENTACIÓN DE OBJETOS 3D**. UV - Informatica. http://informatica.uv.es/iiguia/2000/AIG/web_teoria/tema1.htm

[63] user177800. 23 de Mayo del 2011. ***Re: Why do 3D engines primarily use triangles to draw surfaces?***. [Comentario en el foro *Why do 3D engines primarily use triangles to draw surfaces*?] Stackoverflow. https://stackoverflow.com/questions/6100528/why-do-3d-engines-primarily-use-triangles-to-draw-surfaces

[64] Crawls, Alexa. s.f. ***About Polygon Meshes***. https://web.archive.org/web/20180329092515/http://softimage.wiki.softimage.com:80/xsidocs/poly_basic_PolygonMeshes.htm

[65] ***Coplanaridad***. 9 de Octubre del 2019. Wikipedia. https://es.wikipedia.org/w/index.php?title=Coplanaridad&oldid=120112433

[66] de Vries, Joey. 2014. ***Face culling***. LearnOpenGL. https://learnopengl.com/Advanced-OpenGL/Face-culling

[67] Ongay, Fausto. Julio del 2012. ***Geometría de curvas y computación. 3. Las curvas de Bézier***. [Diapositiva de LaTeX] CIMAT. https://www.cimat.mx/Eventos/veranomma/curvasdeBeizer.pdf

[68] Kenneth I. Joy. 1996. **BERNSTEIN POLYNOMIALS**. On-Line Geometric Modeling Notes. https://web.archive.org/web/20120220143625/http://www.idav.ucdavis.edu/ education/CAGDNotes/Bernstein-Polynomials.pdf

[69] Calvo, Nestor. s.f. ***Curvas y superficies para modelado geométrico***. CIMEC. https://cimec.org.ar/foswiki/pub/Main/Cimec/ComputacionGrafica/curvas.pdf

[70] Drader, J. Scott. Otoño del 2003. ***3D Surface Rendering in Postscript***. https://personal.math.ubc.ca/~cass/courses/m308-03b/projects-03b/drader/main.htm

[71] Holmér, Freya. 16 de Noviembre del 2018. ***The Ever so Lovely Bézier Curve***. Freya Holmér. https://acegikmo.medium.com/the-ever-so-lovely-b%C3%A9zier-curve-eb27514da3bf

[72] Anderson, Scott D. s.f. ***Reading on Bézier Curves and Surfaces***. Wellesley - CS. https://cs.wellesley.edu/~cs307/readings/10-bezier.html

CAPÍTULO IV

Transformaciones geométricas en 2D y 3D

Las *transformaciones geométricas* son funciones que nos permiten mapear un conjunto de vectores a otro, esto al ser multiplicados por una *matriz de transformación* asociada a la transformación deseada. Este tipo de transformaciones son fundamentales en GC, ya que nos permiten modelar objetos, cambiando su posición y/o forma.

En este capítulo se describen las transformaciones geométricas más sencillas que son escalamiento, rotación y traslación, así como el efecto que tienen sobre los diferentes tipos de vectores, como son los vectores de desplazamiento, posición (puntos) y normales.

Se mostrará cómo transformar un conjunto de puntos, esto representando a los puntos como vectores desplazados desde el origen, es decir, los puntos hacen referencia al punto final de los vectores (final de la flecha) con su punto inicial en el origen.

En particular, veremos ejemplos del efecto que tienen dichas transformaciones sobre objetos o modelos (conjunto de puntos) muy simples.

Como se mencionó en el Capítulo 2, los vectores serán representados por vectores columna para el manejo de operaciones. Es importante considerar el tipo de representación a usar, ya que puede afectar tanto al orden en el que se multiplicarían las matrices como al orden de las entradas de éstas.

4.1 Transformaciones geométricas en 2D

Para transformar vectores 2D podemos utilizar matrices 2×2:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Este tipo de operación es una *transformación lineal*, donde se toma a un vector

2D y se obtiene a otro vector 2D como resultado. Con esta simple operación podemos tener una variedad de transformaciones muy útiles.

Otra forma de pensarlo es como una transformación del sistema de coordenadas. Asumiendo que las matrices de transformación son invertibles, vale la pena notar que dicha transformación es parecida a un cambio de base, donde se recibe a un vector en la base descrita por los vectores columna de la matriz, y se obtiene a un vector con sus coordenadas descritas en la base canónica.

4.1.1 Escalamiento

Una transformación de escalamiento como su nombre lo indica, permite cambiar el tamaño de un objeto. En particular, para escalar a un vector, necesitamos multiplicar a cada una de sus coordenadas por un factor de escalamiento, esto es

$$x' = S_x \cdot x \quad \text{y} \quad y' = S_y \cdot y$$

de modo que si $|\ S_x\ | > 1$ se agranda en dirección del eje x, y sí $0 \leq |\ S_x\ | < 1$ se encoge, análogamente con S_y.



Figura 4.1. Escalamiento. Se aplica la matriz $\mathbf{S}(S_x, S_y)$ sobre cada vértice del triángulo.

Se dice que es un escalamiento *uniforme* cuando sus factores de escalamiento son iguales $S_x = S_y$, en otro es un escalamiento *no uniforme*.

Por consiguiente, la matriz de transformación es:

$$\mathbf{S}(S_x, S_y) = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

Puesto que

$$\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} S_x \cdot x \\ S_y \cdot y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Como mencionamos anteriormente, esta operación es parecida a un cambio de base, en este caso es como si tomáramos a un vector representado por una base escalada y lo describiéramos con la base canónica.

4.1.2 Rotación

Supongamos que tenemos un vector $\mathbf{v} = (x, y)$ y lo queremos rotar ϕ grados con respecto al origen, en sentido contrario a las manecillas del reloj, obteniendo $\mathbf{v}' = (x', y')$.

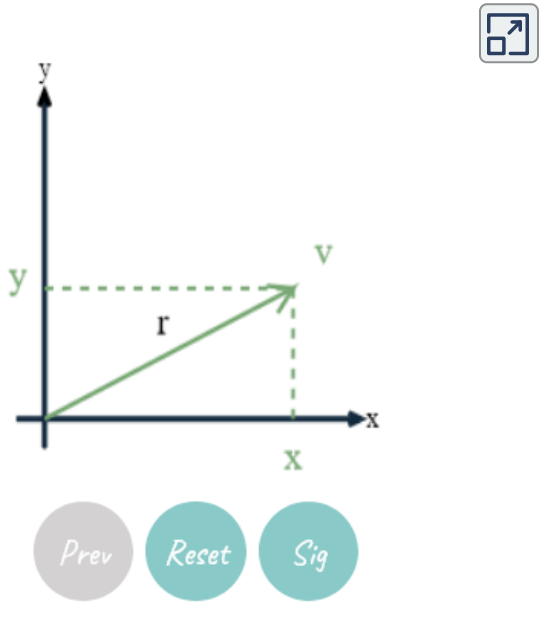


Figura 4.2. Rotación de un vector $\mathbf{v}$.

Observemos que $r = \|\mathbf{v}\|$ y además que entre $\mathbf{v}$ y el eje x se forma un ángulo θ. Entonces, podemos parametrizar a $\mathbf{v}$ como

$$x = r\cos\theta \qquad y = r\sin\theta$$

Y como $\mathbf{v'}$ es $\mathbf{v}$ rotado ϕ grados, su longitud sigue siendo r y termina formando un ángulo de $\theta + \phi$, entonces

$$x' = r\cos(\theta + \phi) \qquad y' = r\sin(\theta + \phi)$$

Sustiuímos por sus identidades trigonométricas, quedando

$$x' = r[\cos\theta\cos\phi - \sin\theta\sin\phi] \qquad y' = r[\sin\theta\cos\phi + \cos\theta\sin\phi]$$

$$\Rightarrow \quad x' = \underbrace{r\cos\theta}_{x}\cos\phi - \underbrace{r\sin\theta}_{y}\sin\phi \qquad y' = \underbrace{r\sin\theta}_{y}\cos\phi + \underbrace{r\cos\theta}_{x}\sin\phi$$

$$\Rightarrow \quad x' = x\cos\phi - y\sin\phi \qquad y' = y\cos\phi + x\sin\phi$$

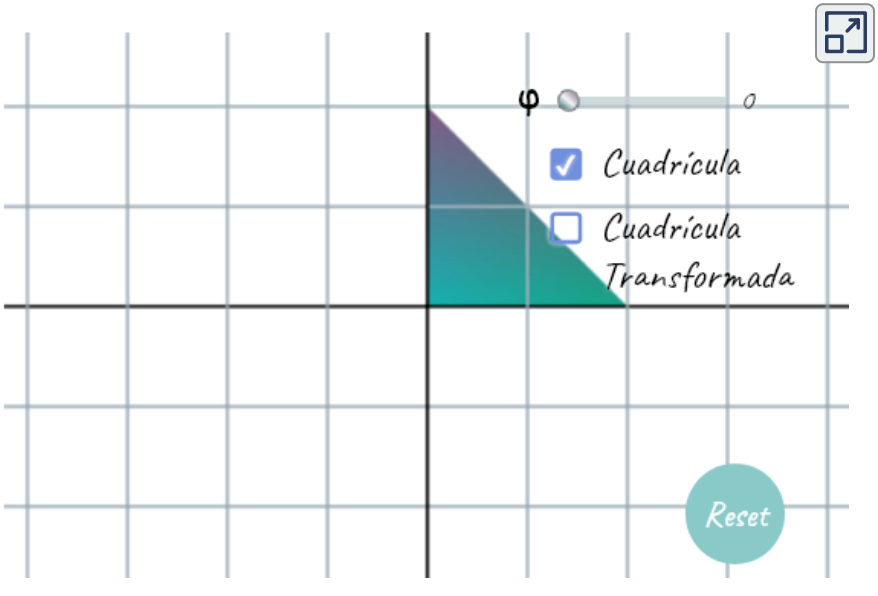


Figura 4.3. Rotación. Se aplica la matriz $\mathbf{R}(\phi)$ sobre cada vértice del triángulo.

Finalmente la matriz de transformación es

$$\mathbf{R}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

Puesto que

$$\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi y + \cos\phi \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Análogamente, en sentido de las manecillas del reloj estaría dada por

$$\mathbf{R}(\phi) = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix}$$

Y de manera similar al escalamiento podemos pensar a los vectores recibidos en una base rotada y finalmente descritos con la base canónica.

4.1.3 Traslación

La transformación de traslación es la más sencilla de todas, ya que basta desplazar ciertas unidades en dirección del eje x o y para generar un movimiento sobre éstos, de modo que

$$x' = x + T_x \quad \text{y} \quad y' = y + T_y$$

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} = \begin{bmatrix} x + T_x \\ y + T_y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

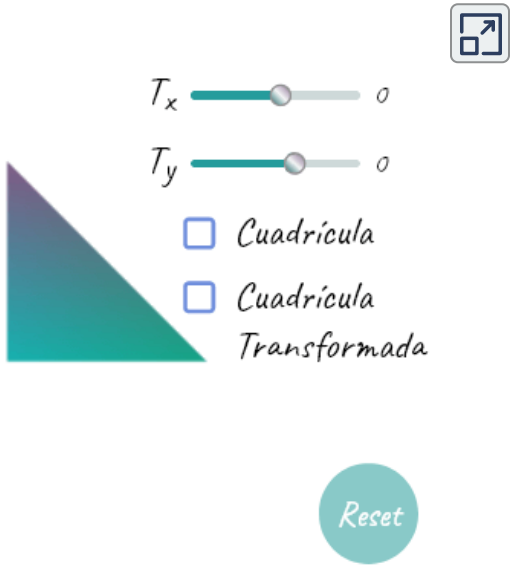


Figura 4.4. Traslación. Se traslada cada vértice del triángulo sumándole el vector (T_x, T_y).

4.2 Transformaciones en coordenadas homogéneas

Con la última transformación podemos notar que para mover un objeto es necesario sumar otro vector a los puntos en lugar de multiplicarlos por una matriz de transformación. Por definición, una *transformación lineal* es una función que preserva la suma y la multiplicación por escalar; en particular, el origen $(0, 0)$ se mantiene fijo. En cambio, con la transformación de traslación vemos que no se satisfacen dichas propiedades.

Esta diferencia hace que se pierda cierta consistencia en el manejo de las transformaciones. Imaginemos que primero queremos escalar y después rotar al punto P_0.

Entonces, primero le aplicaríamos una matriz de escalamiento $\mathbf{S}$, obteniendo

$$P_1 = \mathbf{S}\, P_0,$$

y luego la matriz de rotación $\mathbf{R}$, siendo

$$P_2 = \mathbf{R}\, P_1$$

Recordemos que la multiplicación de matrices es asociativa, entonces

$$P_2 = \mathbf{R}(\mathbf{S}\, P_0) = (\mathbf{RS})\, P_0 = \mathbf{M}\, P_0 \quad \text{donde } \mathbf{M} = (\mathbf{RS})$$

Por lo tanto, podemos representar a estas dos transformaciones con una sola matriz, al multiplicar ambas (de izquierda a derecha), a esto se le conoce como *composición de transformaciones*. Del mismo modo podemos representar a la secuencia de matrices que queramos aplicar como una sola, esto mientras se trate de escalamientos y/o rotaciones (por ahora).

Ya que si queremos aplicar además una traslación quedaría del siguiente modo:

$$P_3 = \mathbf{M}P_0 + \mathbf{T}$$

Pues ya vimos que no hay forma de trasladar objetos haciendo una multiplicación con matrices de 2×2. Por lo que, la expresión empieza a complicarse y se va volviendo menos eficiente. No olvidemos también que es común trabajar con objetos grandes, así como con muchos de ellos, por lo que es más conveniente trabajar con una sola matriz, ya que ésta sería aplicada sobre todos los puntos de los modelos y se reducirían los cálculos.

Con el fin de poder seguir trabajando con esta misma idea, se introdujo el concepto de *coordenadas homogéneas*, donde se agrega una tercera coordenada W a los puntos 2D:

$$P = \begin{bmatrix} x \\ y \\ W \end{bmatrix}$$

De modo que si queremos obtener de vuelta a nuestro punto (en coordenadas Cartesianas), basta dividir las coordenadas sobre W, es decir, $P = (x, y, W) \rightarrow (x/W, y/W)$, con $W \neq 0$.

Como consecuencia, sí dos puntos en coordenadas homogéneas son múltiplos, quiere decir que son iguales, por ejemplo, el punto $(1, 3, 2)$ es igual al punto $(2, 6, 4)$. Sin embargo, se suele utilizar $W = 1$ para los vectores de posición.

Del mismo modo, necesitamos usar matrices de transformación de una dimensión mayor, en este caso, una matriz 3×3, para así poder efectuar la multiplicación con los puntos. Con ello, se puede incluir la traslación en la última columna de la matriz, esto es

$$\mathbf{T}(T_x, T_y) = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

Pues como podemos observar al aplicarla sobre un punto arbitrario P, obtenemos

$$\mathbf{T}\ P = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + T_x \\ y + T_y \\ 1 \end{bmatrix}$$

Mientras que las transformaciones de escalamiento y rotación permanecieron en sus mismas entradas

$$\mathbf{S}(S_x, S_y) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Por otra parte, también necesitamos transformar vectores de dirección o desplazamiento, los cuales no deberían ser afectados por la traslación. Por esta razón, para este tipo de vectores se decidió cambiar a la última componente por un cero, es decir, $W = 0$.

De este modo, tanto la rotación como el escalamiento siguen siendo efectuados, mientras que la traslación es multiplicada por cero e ignorada.

Por lo que ahora podemos representar a una transformación lineal seguida de una traslación con una sola matriz, y a este tipo de transformaciones se les conoce como *transformaciones afín*.



Figura 4.5. Transformaciones afín. Se aplica la matriz $\mathbf{M} = \mathbf{T}(T_x, T_y)\mathbf{R}(\phi)\mathbf{S}(S_x, S_y)$ sobre cada vértice del triángulo.

Rotación y escalamiento con respecto a un punto arbitrario.

Una secuencia de transformaciones muy común y bastante sencilla, es cuando queremos rotar θ grados a un objeto con respecto a un punto arbitrario $Q = (Q_x, Q_y)$ diferente al origen. Por ejemplo, cuando queremos rotarlo sobre su baricentro.

Esta secuencia es la siguiente:

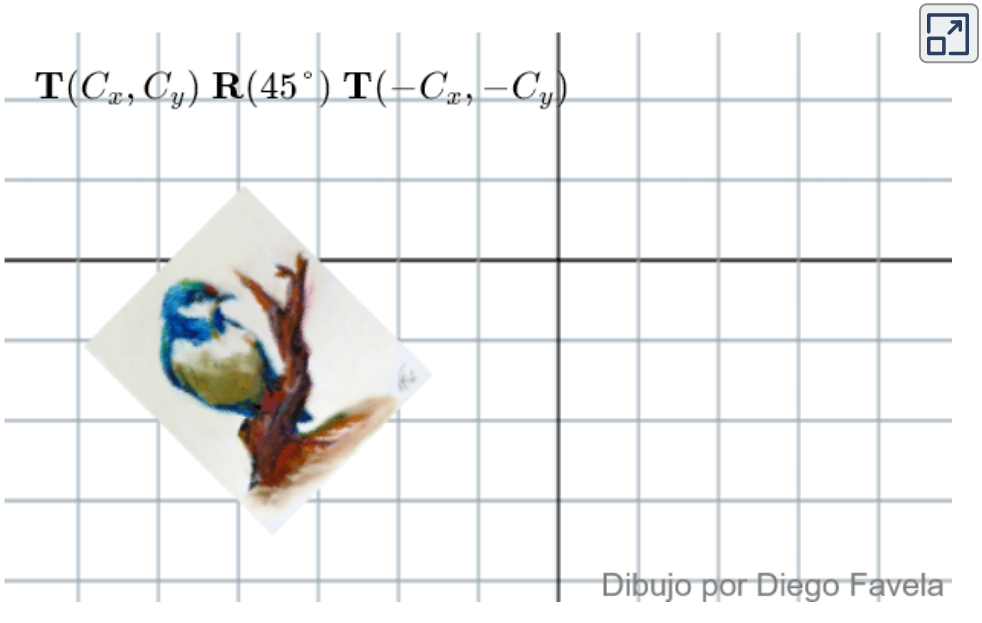
1. Trasladar el objeto $\mathbf{T}(-Q_x, -Q_y)$ de modo que Q quede en el origen.
2. Rotar el objeto θ grados $\mathbf{R}(\theta)$.
3. Trasladar el objeto de regreso $\mathbf{T}(Q_x, Q_y)$ de modo que Q quede en su posición original.

Entonces, la matriz de transformación final está dada por la composición de dichas transformaciones, y se escribiría como

$$\mathbf{T}(Q_x, Q_y)\,\mathbf{R}(\theta)\,\mathbf{T}(-Q_x, -Q_y) = \begin{bmatrix} 1 & 0 & Q_x \\ 0 & 1 & Q_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -Q_x \\ 0 & 1 & -Q_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & -Q_x\cos\theta + Q_y\sin\theta + Q_x \\ \sin\theta & \cos\theta & -Q_x\sin\theta - Q_y\cos\theta + Q_y \\ 0 & 0 & 1 \end{bmatrix}$$

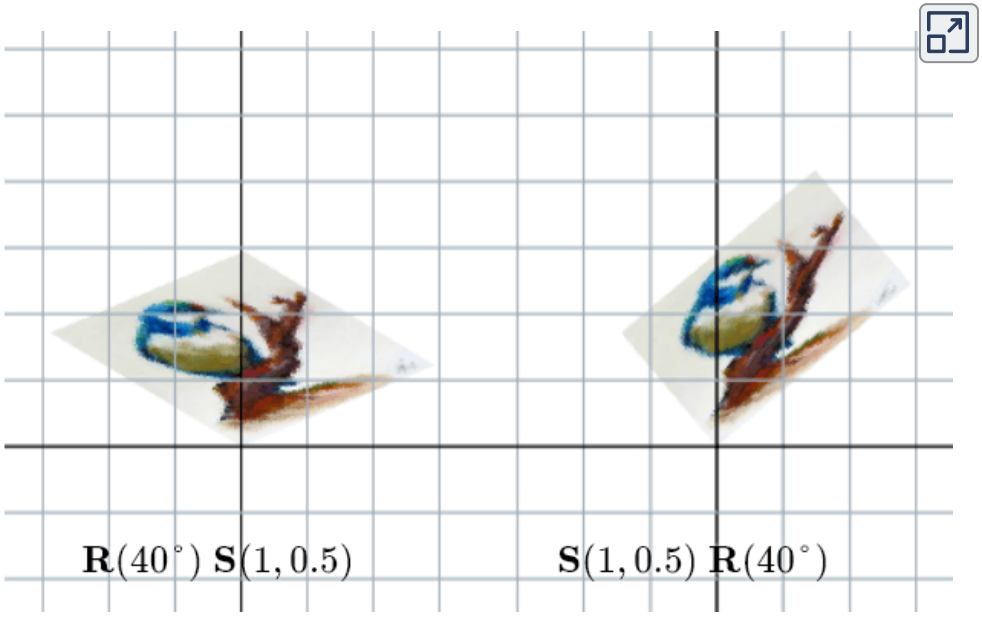
En el siguiente ejemplo se muestra dicha secuencia, rotando un rectángulo sobre su baricentro $C = (C_x, C_y)$.



De manera análoga se haría un escalamiento con respecto a un punto, al simplemente reemplazar la matriz de transformación en el punto 2 por una matriz de escalamiento.

Es importante recordar que la multiplicación de matrices no es conmutativa, por lo que debemos definir bien el orden, de lo contrario el resultado podría ser diferente al que esperamos.

En el siguiente interactivo se puede apreciar mejor este último punto, en donde se presentan dos secuencias muy sencillas, utilizando únicamente dos matrices: $\mathbf{R}(40^\circ)$ y $\mathbf{S}(1, 0.5)$, en distinto orden.



4.3 Transformaciones geométricas en 3D

Las transformaciones geométricas en 3D son una extensión de las 2D. Por lo que, de manera análoga a las transformaciones en 2D, éstas pueden efectuarse a través de una matriz de transformación 4×4.

Ya que, si empezamos analizando el caso de la traslación, podemos notar nuevamente que necesitamos extender nuestras coordenadas, agregando una dimensión más, es decir, utilizando *coordenadas homogéneas*. Por lo que, nuestros vectores de posición serían de la forma $P = (x, y, z, 1)$ y los de des-

plazamiento $\mathbf{d} = (x, y, z, 0)$.

Entonces, la transformación de traslación estaría dada por la transformación

$$x' = x + T_x \quad y' = y + T_y \quad z' = z + T_z$$

$$\text{y su matriz se define como} \quad \mathbf{T}(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Mientras que la de escalamiento a lo largo de los ejes sería

$$x' = S_x \cdot x \quad y' = S_y \cdot y \quad z' = S_z \cdot z$$

$$\mathbf{S}(x, y, z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

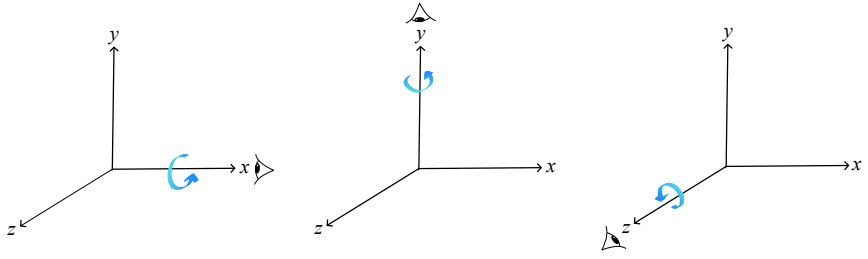
Por otro lado, la transformación de rotación se complica un poco, ya que a diferencia de las transformaciones en 2D (en las que sólo nos preocupábamos por rotar con respecto al origen) tenemos que definir primero con respecto a cuál de los tres ejes se va a rotar.

Cabe recalcar que seguimos tratando con rotaciones positivas, por lo que son en sentido contrario a las manecillas del reloj.

Para la rotación alrededor del eje z, podemos notar que se trata de la misma matriz que en las transformaciones en 2D, al simplemente extenderla para la coordenada z. Pues las coordenadas x y y cambian, mientras que z permanece igual

$$\mathbf{R}(\phi, z) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De manera similar, podemos construir las matrices de rotación alrededor de los ejes x y y descritas a continuación. Podemos pensarlo como si miráramos a lo largo de la mitad positiva del eje de interés con éste en el origen, como se muestra en la **Figura 4.6**.



a) Eje x b) Eje y c) Eje z

Figura 4.6. Rotación sobre un eje 3D. El ojo mira a la mitad positiva del eje de interés.

Por lo que, para la rotación alrededor del eje y

$$z' = z\cos\phi - x\sin\phi \quad x' = x\cos\phi + z\sin\phi \quad y' = y$$

$$\mathbf{R}(\phi, y) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finalmente para la rotación con respecto al eje x

$$y' = y\cos\phi - z\sin\phi \quad z' = z\cos\phi + y\sin\phi \quad x' = x$$

$$\mathbf{R}(\phi, x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Un punto importante es que todas las matrices de transformación que hemos visto hasta ahora son invertibles. Pues podemos deducir de manera intuitiva que la inversa de la transformación de traslación es la traslación misma en dirección contraria. O bien, que la inversa de una matriz de escalamiento está dada por $\mathbf{S}(1/s_x, 1/s_y, 1/s_z)$.

Por último, podemos invertir todas las matrices de rotación al simplemente hacer la rotación en sentido contrario, o aprovechando que son matrices ortogonales, con su transpuesta.

Y ya que podemos definir a una matriz de transformación como una composición de matrices $\mathbf{M} = \mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n$, entonces podemos obtener

su inversa como $\mathbf{M}^{-1} = \mathbf{M}_n^{-1}, \ldots, \mathbf{M}_2^{-1}, \mathbf{M}_1^{-1}$, es decir, multiplicando sus matrices inversas en orden contrario.

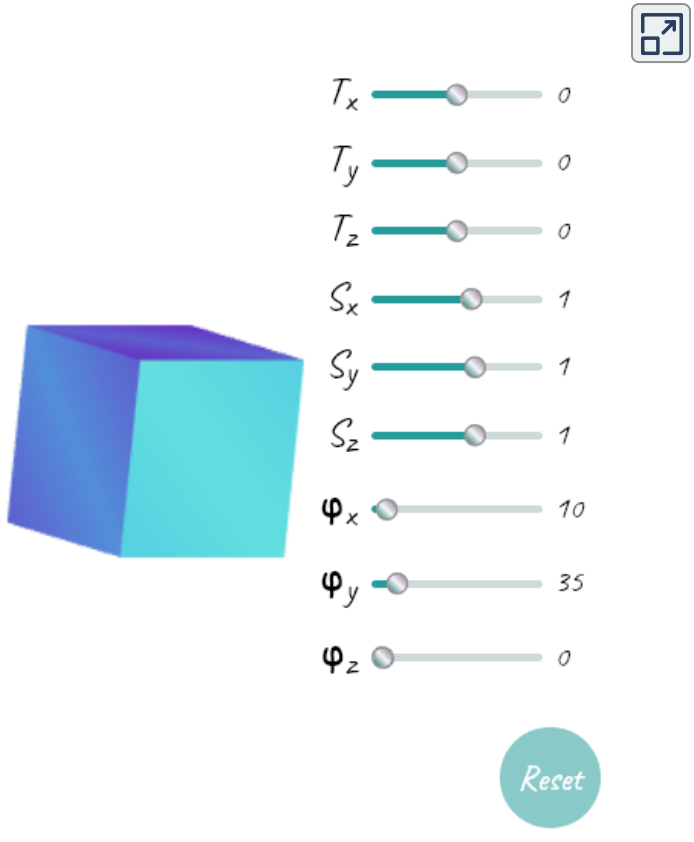


Figura 4.7. Transformaciones 3D. Se aplica la matriz $\mathbf{M} = \mathbf{T}(T_x, T_y)\mathbf{R}(\phi, z)\mathbf{R}(\phi, y)\mathbf{R}(\phi, x)\mathbf{S}(S_x, S_y)$ sobre cada vértice del cubo.

Rotación sobre un eje arbitrario.

Digamos que queremos rotar a un vector $\mathbf{v}$ en un ángulo θ alrededor de un eje arbitrario cuya dirección está dada por un vector unitario $\mathbf{a}$. Para ello, podemos descomponer a $\mathbf{v}$ como la suma de dos vectores $\mathbf{v}_\parallel$ y $\mathbf{v}_\perp$, siendo el vector paralelo y el vector perpendicular a $\mathbf{a}$ respectivamente, es decir, $\mathbf{v} = \mathbf{v}_\parallel + \mathbf{v}_\perp$.

De modo que si rotamos a ambos vectors alrededor de $\mathbf{a}$ podemos obtener $\mathbf{v}'$ nuevamente como $\mathbf{v}' = \mathbf{v}'_\parallel + \mathbf{v}'_\perp$. Y como $\mathbf{v}_\parallel$ no cambia durante la rotación, entonces $\mathbf{v}'_\parallel = \mathbf{v}_\parallel$, reduciendo el problema a encontrar solo a $\mathbf{v}'_\perp$.

Como $\mathbf{v}_{\parallel}$ es la proyección de $\mathbf{v}$ sobre $\mathbf{a}$, y además $\mathbf{a}$ es un vector unitario tenemos que

$$\mathbf{v}_{\parallel} = (\mathbf{v} \cdot \mathbf{a})\mathbf{a}$$

Por otra parte, como $\mathbf{v} = \mathbf{v}_{\parallel} + \mathbf{v}_{\perp}$ entonces podemos calcular al vector perpendicular como

$$\mathbf{v}_{\perp} = \mathbf{v} - (\mathbf{v} \cdot \mathbf{a})\mathbf{a}$$

Ahora solo hace falta rotar a $\mathbf{v}_{\perp}$ sobre $\mathbf{a}$ para obtener a $\mathbf{v}'$. Notemos que $\mathbf{v}'$ se encuentra sobre el plano perpendicular al eje $\mathbf{a}$, por lo que podemos expresarlo como una combinación lineal de $\mathbf{v}_{\perp}$ y el vector obtenido de rotar 90° a $\mathbf{v}_{\perp}$ sobre $\mathbf{a}$, el cual podemos obtener fácilmente con $(\mathbf{a} \times \mathbf{v}_{\perp})$. Pudiendo así obtener a $\mathbf{v}'_{\perp}$ como la combinación lineal de estos dos vectores base

$$\mathbf{v}'_{\perp} = \cos\theta \mathbf{v}_{\perp} + \sin\theta(\mathbf{a} \times \mathbf{v}_{\perp})$$

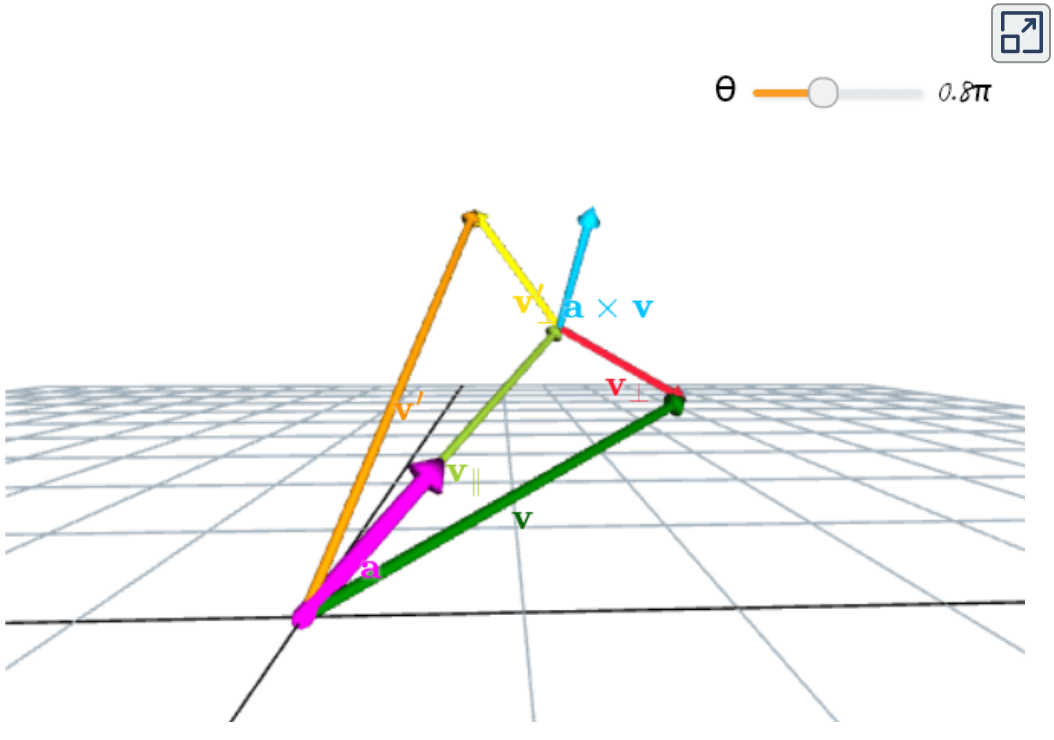


Figura 4.8. Rotación sobre un eje arbitrario. Se invita al lector a modificar el valor del ángulo de rotación θ, así como cambiar la posición de la cámara.

$$\text{Y como} \quad \begin{aligned}\mathbf{a} \times \mathbf{v}_{\perp} &= \mathbf{a} \times (\mathbf{v} - \mathbf{v}_{\parallel}) = \mathbf{a} \times \mathbf{v} - \mathbf{a} \times \mathbf{v}_{\parallel} \\ &= \mathbf{a} \times \mathbf{v} - \mathbf{0} = \mathbf{a} \times \mathbf{v}\end{aligned}$$

Entonces podemos obtener al vector rotado $\mathbf{v}'$ como la suma

$$\begin{aligned}\mathbf{v}' &= (\mathbf{v} \cdot \mathbf{a})\mathbf{a} + \cos\theta(\mathbf{v} - (\mathbf{a} \cdot \mathbf{v})\mathbf{a}) + \sin\theta(\mathbf{a} \times \mathbf{v}) \\ &= (1 - \cos\theta)(\mathbf{v} \cdot \mathbf{a})\mathbf{a} + \cos\theta\mathbf{v} + \sin\theta(\mathbf{a} \times \mathbf{v})\end{aligned}$$

Los vectores se muestran en la **Figura 4.8**.

Para construir la matriz de transformación, llamémosla $\mathbf{R}(\theta, \mathbf{a})$, empezamos por obtener las matrices equivalentes de los vectores en la ecuación de $\mathbf{v}'$

$$(\mathbf{v} \cdot \mathbf{a})\mathbf{v} = (p_x a_x + p_y a_y + p_x a_y) \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z \\ a_x a_y & a_y^2 & a_y a_z \\ a_x a_z & a_y a_z & a_z^2 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$(\mathbf{a} \times \mathbf{v}) = \begin{bmatrix} a_y p_z - a_z p_y \\ a_z p_x - a_x p_z \\ a_x p_y - a_y p_x \end{bmatrix} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Por último, sustituimos las matrices en la ecuación. Sea $s = \sin\theta$ y $c = \cos\theta$

$$\mathbf{v}' = (1 - \cos\theta)(\mathbf{v} \cdot \mathbf{a})\mathbf{a} + \cos\theta\mathbf{v} + \sin\theta(\mathbf{a} \times \mathbf{v})$$

$$= (1 - c) \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z \\ a_x a_y & a_y^2 & a_y a_z \\ a_x a_z & a_y a_z & a_z^2 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + c \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + s \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$= \left[(1 - c) \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z \\ a_x a_y & a_y^2 & a_y a_z \\ a_x a_z & a_y a_z & a_z^2 \end{bmatrix} + c \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + s \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \right] \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Finalmente

$$\mathbf{R}(\theta, \mathbf{a}) = \begin{bmatrix} (1 - c)a_x^2 + c & (1 - c)a_x a_y - s a_z & (1 - c)a_x a_z + s a_y & 0 \\ (1 - c)a_x a_y + s a_z & (1 - c)a_y^2 + c & (1 - c)a_y a_z - s a_x & 0 \\ (1 - c)a_x a_z - s a_y & (1 - c)a_y a_z + s a_x & (1 - c)a_z^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.4 Transformación de la normal

Un vector es un *vector normal* si es perpendicular al plano tangente de una superficie, en particular, a los vectores que sean tangentes a la superficie.

Cuando transformamos un objeto con una matriz $\mathbf{M}$, necesitamos transformar no solo sus posiciones si no también sus vectores normales.

Por un lado, los vectores tangentes pueden ser calculados tomando la diferencia entre los vértices que conforman la superficie, entonces, también podríamos obtener los nuevos vectores tangentes transformados como la diferencia de los nuevos vértices transformados. Por lo que la misma matriz $\mathbf{M}$, puede usarse para obtener los nuevos vectores tangentes correctamente.

Pero para los vectores normales no es tan simple, ya que si usamos una matriz *no ortogonal* la normal transformada puede terminar apuntando a una dirección que no es perpendicular a la nueva superficie transformada (Ver **Figura 4.9**).

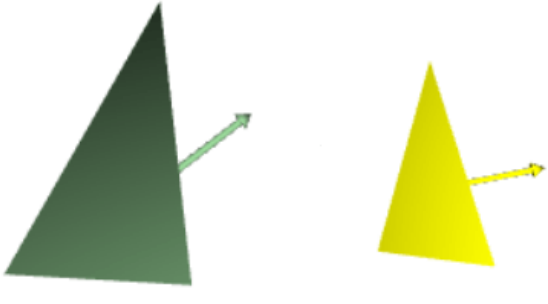


Figura 4.9. Ejemplo de la transformación de la normal. La pirámide de la izquierda (verde) muestra lo que ocurre cuando se usa una matriz no ortogonal para transformar al vector normal, el cual termina apuntando en una dirección que ya no es perpendicular a la superficie transformada, mientras que la pirámide de la derecha (amarilla) muestra cómo debería permanecer dicha normal, es decir, siempre perpendicular a la superficie. También podemos observar que el sombreado resulta afectado por la normal, en especial sobre los picos de las pirámides.

Ahora bien, como el vector tangente $\mathbf{t}$ y el vector normal $\mathbf{n}$ deben ser perpendiculares se satisface que su producto punto es cero, lo cual puede expresarse como el producto de dos matrices $\mathbf{n} \cdot \mathbf{t} = \mathbf{n}^T\mathbf{t} = 0$.

Por lo que necesitamos que esta misma ecuación se siga cumpliendo para el vector tangente transformado $\mathbf{t}'$ y la normal transformada $\mathbf{n}'$.

Sabemos que $\mathbf{t}' = \mathbf{M}\mathbf{t}$, entonces necesitamos encontrar una matriz $\mathbf{N}$ tal que

$$\mathbf{n}'^T\mathbf{t}' = (\mathbf{N}\mathbf{n})^T(\mathbf{M}\mathbf{t}) = 0$$

Si desarrollamos un poco más tenemos que

$$(\mathbf{N}\mathbf{n})^T(\mathbf{M}\mathbf{t}) = \mathbf{n}^T\mathbf{N}^T\mathbf{M}\mathbf{t}$$

Por otro lado, como sabemos que $\mathbf{n}^T\mathbf{t} = 0$, entonces podemos agregar a la matriz identidad dentro del producto punto y tomar ventaja de que $\mathbf{I} = \mathbf{M}^{-1}\mathbf{M}$

$$\mathbf{n}^T\mathbf{t} = \mathbf{n}^T\mathbf{I}\mathbf{t} = \mathbf{n}^T\mathbf{M}^{-1}\mathbf{M}\mathbf{t} = 0$$

Así pues, es claro ver que la ecuación para los vectores transformados solo se satisface sí $\mathbf{N}^T\mathbf{M} = \mathbf{I}$, por consiguiente $\mathbf{N} = (\mathbf{M}^{-1})^T$, es decir, los vectores normales pueden ser correctamente transformados con la transpuesta de la inversa de la matriz que se use para transformar los puntos.

Nótese que si la matriz de transformación es *ortogonal* entonces $(\mathbf{M}^{-1})^T = (\mathbf{M}^T)^T = \mathbf{M}$.

4.5 Referencias bibliográficas

[73] Shirley, Peter, et al. ***Fundamentals of Computer Graphics.*** A K Press. 3ª Edición. 2009.

[74] Klawonn, Frank. ***Introduction to Computer Graphics Using Java 2D and 3D.*** Springer-Verlag. 2ª Edición. 2012.

[75] Ganovelli, Fabio, et al. ***Introduction to Computer Graphics a Practical Learning Approach.*** CRC Press. 2015.

[76] Shalini Govil-Pai. ***Principles Of Computer Graphics Theory and Practice Using OpenGL and Maya.*** Springer. 1ª Edición. 2005.

[77] Boreskov, Alexey, et al. ***Computer Graphics From pixels to programmable Graphics hardware.*** CRC Press. 2014.

[78] Dunn, Fletcher, et al. ***3D Math Primer for Graphics and Game Development.*** CRC Press. 2ª Edición. 2011.

[79] ***W3schools***. https://www.w3schools.com/

[80] ***Htmlcolorcodes***. https://htmlcolorcodes.com/es/

[81] ***WebGL2 2D Translation***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-2d-translation.html

[82] Tavares, Gregg. ***TWGL: A Tiny WebGL helper Library***. https://twgljs.org/docs/

[83] Tavares, Gregg. ***TWGL: A Tiny WebGL helper Library***. Github. https://github.com/greggman/twgl.js/

[84] ***WebGL2 Shaders and GLSL***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-shaders-and-glsl.html

[85] ***WebGL2 - Less Code, More Fun***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-less-code-more-fun.html

[86] ***WebGL2 - Animation***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-animation.html

[87] ***WebGL2 2D Matrices***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-2d-matrices.html

[88] ***WebGL2 Multiple Views, Multiple Canvases***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-multiple-views.html

[89] ***Variable scope, closure***. JAVASCRIPT.INFO. Recuperado 8 de Agosto del 2020 de https://javascript.info/closure

[90] ***WebGL2 - Drawing Multiple Things***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-drawing-multiple-things.html

[91] Ferdinandi, Chris. ***Debouncing events with requestAnimationFrame() for better performance***. GoMakeThings. 11 de Octubre del 2017. https://gomakethings.com/debouncing-events-with-requestanimationframe-for-better-performance/

[92] Ho Ahn, Song. 2005-2018. ***OpenGL Rotation About Arbitrary Axis***. Songho.ca. http://www.songho.ca/opengl/gl_rotate.html

[93] Belousov, Boris. 31 de Mayo del 2016. ***Change of basis vs linear transformation***. Boris Belousov. http://boris-belousov.net/2016/05/31/change-of-basis/

[94] BananaCoding. 4 de Abril del 2019. ***JavaScript + CSS Range Slider | Costum Value Range Slider Webdesign Tutorial***. [Archivo de Vídeo]. Youtube. https://youtu.be/BrpiNUf2XCk/

CAPÍTULO V

Transformaciones de vista en 3D

Como vimos en el Pipeline gráfico, en el procesamiento geométrico, una vez que hayamos hecho las transformaciones (rotación, traslación, escalamiento) deseadas para así modelar a los objetos dentro un espacio 3D sigue aplicar las transformaciones de vista, las cuales constan de una secuencia de tres transformaciones: la transformación de la cámara, de proyección y de *viewport*.

5.1 Transformación de la cámara

La transformación de la cámara transforma a todos los puntos definidos en el espacio global (incluyendo la cámara misma), al *espacio de la cámara* o *vista*, donde la cámara queda posicionada en el origen $(0, 0, 0)$ para simplificar los cálculos, y queda mirando hacia el eje $z-$ *negativo* por convención. Esto es

$$\mathbf{M}_{\text{cam}} \underbrace{\begin{bmatrix} x_{global} \\ y_{global} \\ z_{global} \\ w_{global} \end{bmatrix}}_{\text{coordenadas globales}} = \mathbf{M}_{\text{cam}}\mathbf{M}_{\text{model}} \underbrace{\begin{bmatrix} x_{objeto} \\ y_{objeto} \\ z_{objeto} \\ w_{objeto} \end{bmatrix}}_{\text{coordenadas del objeto}} = \underbrace{\begin{bmatrix} x_{vista} \\ y_{vista} \\ z_{vista} \\ w_{vista} \end{bmatrix}}_{\text{coordenadas de la vista}}$$

donde $\mathbf{M}_{\text{model}}$ es la matriz de transformación del objeto como mencionamos antes, llamada *Model matrix*.

Mientras que para las normales sería

$$((\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{model}})^{-1})^{T} \underbrace{\begin{bmatrix} x_{objeto} \\ y_{objeto} \\ z_{objeto} \\ w_{objeto} \end{bmatrix}}_{\text{coordenadas del objeto}} = \underbrace{\begin{bmatrix} x_{vista} \\ y_{vista} \\ z_{vista} \\ w_{vista} \end{bmatrix}}_{\text{coordenadas de la vista}}$$

Pues como vimos la transformación de la normal se calcula diferente.

La transformación de vista nos permite definir la orientación de la cámara, es decir, su posición y a dónde estará mirando.

Es importante hacer notar que desde las coordenadas globales se ha estado usando un *sistema de coordenadas de mano derecha*, pues por convención las transformaciones de rotación, traslación y escalamiento están especificadas de este modo. Esto es que si consideramos a la pantalla como el plano xy, entonces el eje $x+$ se encontraría apuntando a la derecha, el eje $y+$ hacia arriba y el eje $z+$ saldría de la pantalla.

Supongamos entonces que queremos posicionar la cámara en el vector de posición $\mathbf{e}$ (ojo) y que además esté apuntando hacia el objetivo $\mathbf{g}$, con un vector $\mathbf{t}$ el cual señale la dirección hacia arriba de la cámara. Estos vectores nos dan la información suficiente para construir un sistema de coordenadas con la base $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ con el origen en $\mathbf{c}$, donde

$$\mathbf{w} = \frac{(\mathbf{e} - \mathbf{g})}{\|\mathbf{e} - \mathbf{g}\|}$$

$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\|\mathbf{t} \times \mathbf{w}\|}$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

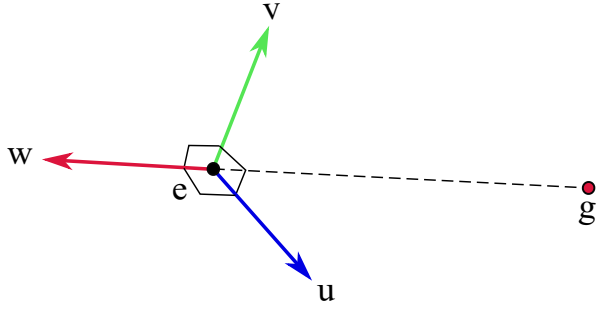


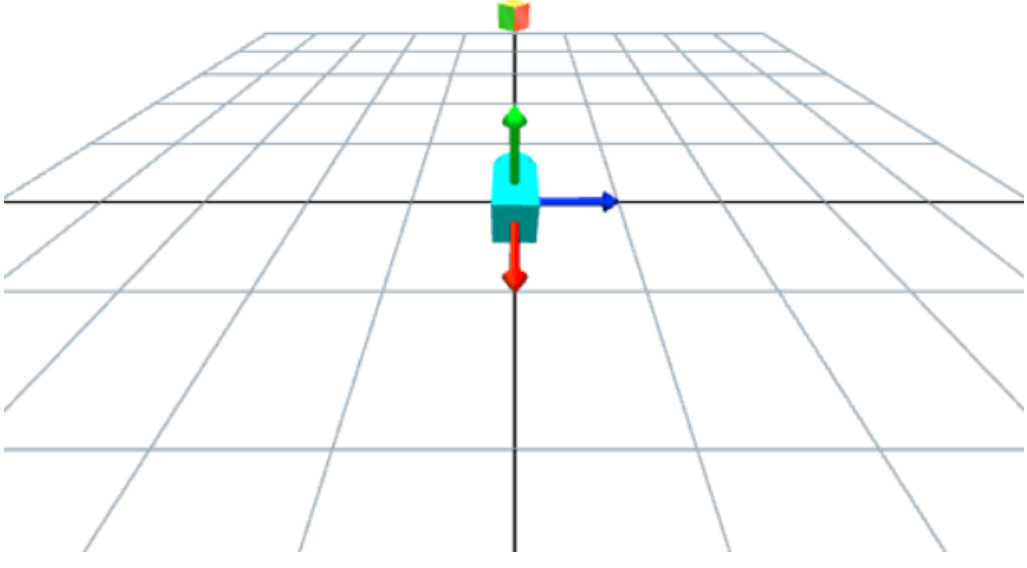
Figura 5.1. Vectores $\mathbf{u}$, $\mathbf{v}$ y $\mathbf{w}$ del objetivo a la cámara.

Ahora bien, tomando en cuenta que los objetos están en coordenadas globales, con origen en $(0, 0, 0)$ y los ejes $\mathbf{x}$, $\mathbf{y}$ y $\mathbf{z}$ como base (base canónica). Y ya que queremos transformarlos al espacio de la vista definido a partir de la base

recién definida $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$, podemos construir la transformación de la vista con dos transformaciones: una traslación, que mueve la cámara al origen, y un cambio de base, o bien, una rotación para orientar los ejes de la cámara para que se alinee el vector $\mathbf{u}$ con el eje $\mathbf{x}$, a $\mathbf{v}$ con el eje $\mathbf{y}$ y a $\mathbf{w}$ con $\mathbf{z}$.

Entonces podemos construir la matriz de transformación de la cámara como

$$\mathbf{M}_{\text{cam}} = \underbrace{\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{cambio de base}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{traslación}} = \begin{bmatrix} u_x & u_y & u_z & -e \cdot u \\ v_x & v_y & v_z & -e \cdot v \\ w_x & w_y & w_z & -e \cdot w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Una manera de recordar cómo van posicionados cada uno de los vectores base es como sigue. Como queremos que $\mathbf{u}$ se vuelva $\mathbf{x}$, es decir, el vector $(1, 0, 0)$, podemos ver que al multiplicar la matriz de cambio de base con $\mathbf{u}$, obtenemos a $\mathbf{x}$, pues $\mathbf{u} \cdot \mathbf{u} = 1$. En cambio la multiplicación del segundo y tercer renglón debería ser igual a $\mathbf{0}$, ya que son ortogonales a $\mathbf{u}$. Y de manera similar con $\mathbf{v}$ y $\mathbf{w}$ podemos obtener a $\mathbf{y}$ y $\mathbf{z}$ respectivamente.

5.2 Transformación de proyección

La transformación de proyección transforma los puntos definidos en el espacio de la vista al espacio de *clip* o recorte, el cual está definido por un volumen de visión dado por el tipo de proyección a utilizar, de modo que los puntos que se encuentren dentro éste serán los puntos que se verán en la pantalla. Esto es

$$\mathbf{M}_{\text{proy}} \underbrace{\begin{bmatrix} x_{vista} \\ y_{vista} \\ z_{vista} \\ w_{vista} \end{bmatrix}}_{\text{coordenadas de la vista}} = \underbrace{\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix}}_{\text{coordenadas de clipping}}$$

Por otro lado, recordemos que estamos utilizando todo el tiempo coordenadas homogeneas, con su última componente $W = 1$ para los puntos. Y en particular que el vector homogéneo (x, y, z, W) representa al punto $(x/W, y/W, z/W)$. Pues esta última operación, donde se realiza la división entre la componente W, hace referencia a la siguiente transformación.

Si bien, esto no produce ningún cambio cuando $W = 1$, que es el caso de la proyección ortográfica, es necesaria para la proyección de perspectiva, por esta razón es llamada *división de perspectiva*. Al ser una operación demasiado simple y depender de la transformación de proyección que se decida, decidimos dejarla aquí.

De modo que antes de pasar a la última proyección de vista, las coordenadas de recorte o *clipping* son transformadas a coordenadas normalizadas del dispositivo o NDC por sus siglas en inglés, con las que se asegura estar dentro del volumen canónico de vista, esto es

$$\underbrace{\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix}}_{\text{coordenadas de clipping}} \Rightarrow \begin{bmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{bmatrix} = \underbrace{\begin{bmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{bmatrix}}_{\text{coordenadas NDC}}$$

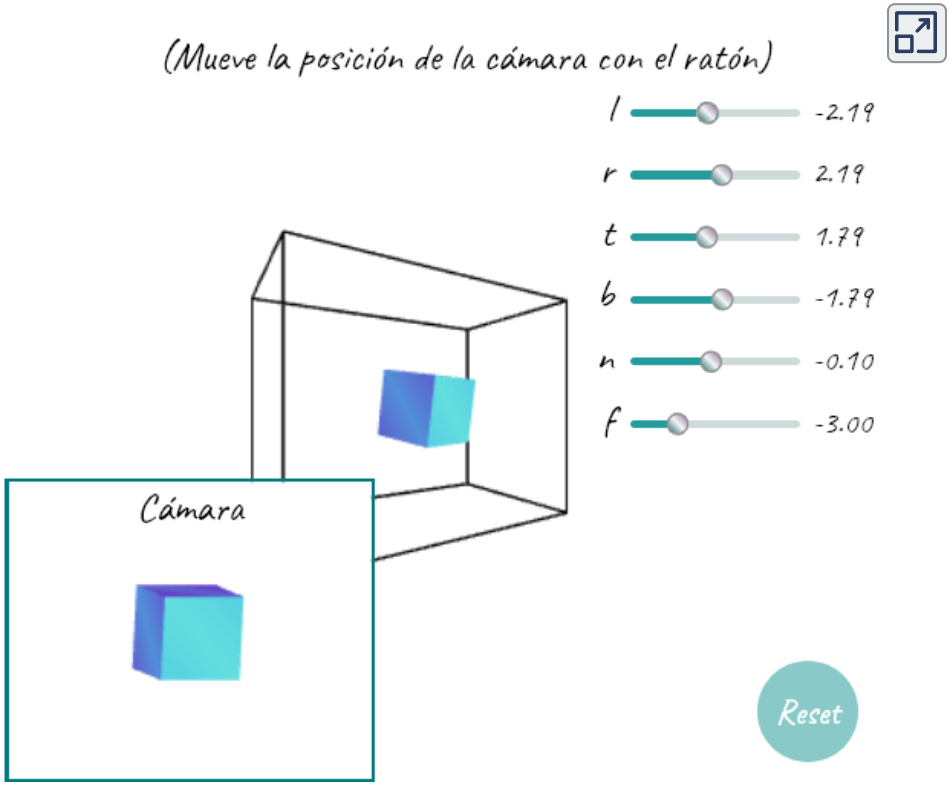
En otras palabras, la transformación de proyección define el volumen de visión que la cámara estará viendo, y que será transformado al volumen canónico de vista.

El *volumen canónico de vista* que consideraremos es un cubo de lado 2 centrado en el origen. Por lo que las coordenadas NDC estarán dentro del cubo $[-1, 1]^3$.

Aunado a esto, para las transformaciones de proyección se seguirá usando un sistema de coordenadas de mano derecha, y se asume que la cámara está en el origen mirando a lo largo dej eje $z-$.

5.2.1 Transformación de proyección ortográfica

Una característica de la proyección ortográfica es que las líneas paralelas se mantienen paralelas después de la proyección. Por lo que, cuando se usa, los objetos preservan su tamaño independientemente de su distancia con la cámara.



La proyección ortográfica define a su volumen de visión como una caja rectangular, también llamada *axis-aligned box*, formada por los planos: (l) izquierdo, (r) derecho, (b) inferior, (t) superior, (n) cercano o frontal y (f) lejano.

Notemos que como la cámara está mirando hacia $z-$ tenemos que $n > f$, ya que el plano n es el más cercano a la cámara y f sería un número negativo más grande, es decir, más pequeño que n.

Entonces, la transformación de proyección ortográfica mapea un punto dentro del paralelepípedo $[l, r] \times [b, t] \times [n, f]$ al volumen canónico de vista, es decir, mapeamos la coordenada x del punto que está dentro de un rango $[l, r]$ a $[-1, 1]$, de manera similar la coordenada y de $[b, t]$ a $[-1, 1]$ y la coordenada z de $[-n, -f]$ a $[-1, 1]$.

Una forma de obtener dicha transformación es simplemente trasladar al paralelepípedo de modo que su centro quede en el origen, y después escalarlo al tamaño del volumen canónico de vista, esto último con la proporción entre las dimensiones del cubo y paralelepípedo, teniendo

$$\mathbf{M}_{\text{orth}} = \underbrace{\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{escalamiento}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{traslación}}$$

$$= \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{n-f} & -\frac{f+n}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

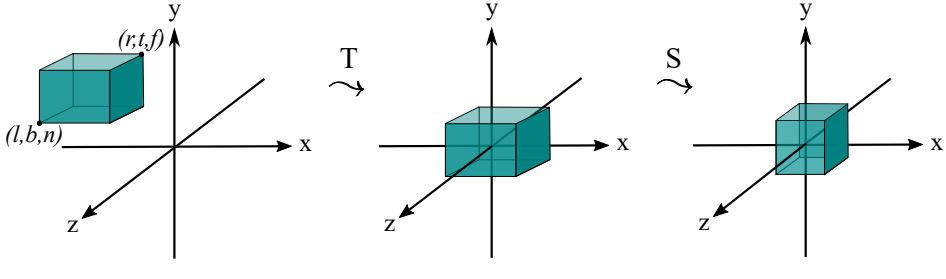


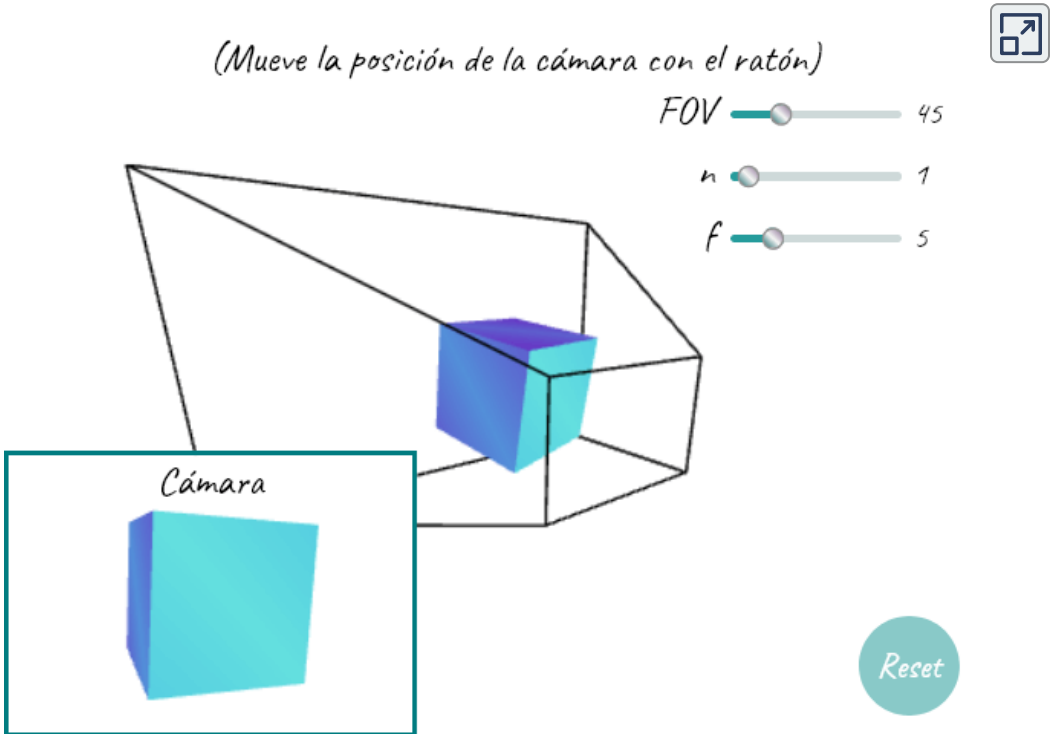
Figura 5.2. Transformación del volumen de visión ortográfico al volumen canónico de vista.

O bien, podemos pensarlo como una transformación de ventana. Y ya que está transformación es efectuada con una traslación y escalamiento, es invertible.

A pesar de que muchos de los APIs definen al volumen canónico con estos mismos límites, como OpenGL, hay otras propuestas que mapean la coordenada z en un rango $[0, 1]$.

5.2.2 Transformación de proyección de perspectiva

A diferencia de la proyección ortográfica, en la proyección de perspectiva las líneas paralelas no suelen permanecer paralelas después de la proyección, y es un poco menos simple. Esta proyección suele ser la más utilizada en GC ya que se ajusta más a la manera en que percibimos el mundo, es decir, los objetos más lejanos se visualizan más pequeños y viceversa.



Y para lograr esta sensación de profundidad se utiliza el mismo principio que en la cámara pinhole.

La proyección de perspectiva mapea las coordenadas x y y de cada punto dentro del volumen de visión, el cual tiene forma de pirámide truncada o frustrum, a un punto en el plano de proyección, guardando a su vez la posición de la coordenada z.

Para esta proyección seguiremos considerando que la cámara está posicionada en el origen, mirando hacia $z-$. Así como a los planos n (cercano o frontal) y f (lejano) que delimitan al volúmen de visión.

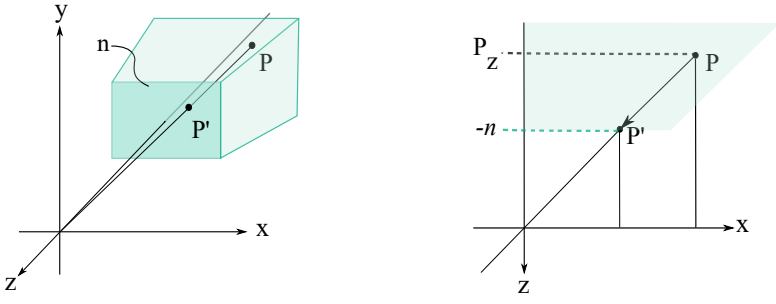


Figura 5.3. Proyección P' de un punto P sobre el plano de proyección n.

En particular, usaremos al plano n como el plano de proyección (*image plane*), teniendo entonces una distancia focal de n. Por lo que, al igual que con la cámara pinhole, podemos calcular las coordenadas del punto proyectado usando la razón de los triángulos semejantes formados entre el punto en el volumen P y el proyectado P'

$$x' = -\frac{n}{z}x \quad \text{y} \quad y' = -\frac{n}{z}y$$

Sin embargo este tipo de transformación, en donde una de las coordenadas del vector aparece como común denominador, no puede lograrse con las transformaciones afines. Por otro lado, como mencionamos antes, al multiplicar la matriz de proyección sobre las coordenadas de vista, éstas son transformadas a coordenadas de recorte, las cuales siguen siendo coordenadas homogeneas. De modo que para mapear dichas coordenadas al volumen canónico de vista o coordenadas NDC, son divididas por su componente W.

Por lo tanto, podemos obtener a los puntos proyectados haciendo: $W = -z$, esto es, poner el último renglón de la matriz de proyección como $(0, 0, -1, 0)$, y escalando cada componente por la distancia del plano n.

Ahora solo bastaría calcular de manera similar que con la proyección ortográfica, el mapeo de las coordenadas proyectadas x' y y' a los rangos del volumen canónico, donde l, r, b y t delimitan al plano de proyección n, siendo (l, b, n) la esquina izquierda inferior y (r, t, n) la esquina superior derecha del rectángulo. Dicho de otra forma, la coordenada x' en $[l, r]$ se mapea $[-1, 1]$, y la

coordenada y de $[b, t]$ a $[-1, 1]$. Teniendo hasta ahora a la matriz de perspectiva

$$\mathbf{M}_{\text{persp}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2n}{t-b} & 0 & -\frac{t+b}{t-b} \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Por lo que, al igual que con la cámara pinhole, los límites (l, r) definirían el ángulo de visión horizontal de la cámara, y (t, b) el ángulo de visión vertical.

Por otra parte la coordenada z_{ndc} se calcula diferente, ya que si tomamos la coordenada z' del punto proyectado, ésta siempre sería $-n$. Por lo que, necesitamos "desproyectar" a z', siendo más específicos, su posición entre los planos n y f. Pues más adelante necesitaremos este valor para hacer comparaciones de profundidad tanto en el *clipping* como en el algoritmo de z-búfer. Además de que así podremos invertir la matriz de proyección.

Para ello, podemos modificar el tercer renglón de la matriz de proyección, de modo que obtengamos a z, y que además esté en un rango de $[-1, 1]$. Entonces podemos expresar a la coordenada z_{ndc} de la forma

$$z_{ndc} = -\frac{Az + B}{z}$$

Y para resolver los coeficientes A y B, podemos usar la relación de $[-n, -f] \rightarrow [-1, 1]$, esto es

$$\begin{cases} -(An + B)/n = -1 \\ -(Af + B)/f = 1 \end{cases} \rightarrow \begin{cases} -An + B = -n \\ -Af + B = f \end{cases}$$

Entonces

$$\begin{cases} \quad -An + B = -n \\ -\ \underline{-Af + B = f} \\ A(-n + f) = -n - f \end{cases} \Rightarrow A = -\frac{f + n}{f - n}$$

Ahora, sustituimos en la primera ecuación

$$\left(\frac{f + n}{f - n}\right) n + B = -n$$

$$\Rightarrow B = -n - \left(\frac{f+n}{f-n}\right)n$$

$$\begin{aligned} B &= -\left(1 + \frac{f+n}{f-n}\right)n \\ &= -\left(\frac{(f-n)+(f+n)}{f-n}\right)n = -\left(\frac{2f}{f-n}\right)n \\ &= -\frac{2fn}{f-n} \end{aligned}$$

Por lo tanto

$$z_{ndc} = \left(-\frac{f+n}{f-n}z - \frac{2fn}{f-n}\right)/z$$

Finalmente la matriz de proyección de perspectiva sería

$$\mathbf{M}_{\text{persp}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2n}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

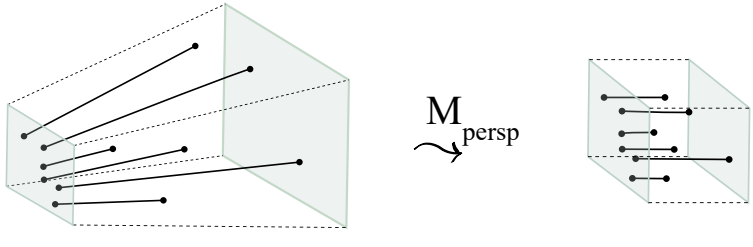


Figura 5.4. Transformación del frustrum al volumen canónico de vista.

Notemos que la relación entre la coordenada z_{vista} y z_{ndc} no es lineal, de manera que mientras más cercano se encuentre un punto al plano cercano n, su distancia será más precisa, en caso contrario, será menos precisa cuando se acerque al plano lejano f. Si bien, no es posible obtener la coordenada original,

una forma de minimizar el problema es posicionar a los planos lo más cerca posible.

Es posible también construir una matriz de perspectiva que al igual que una cámara real tenga un campo de profundidad infinito, haciendo que la distancia del plano f tienda a infinito

$$\mathbf{M}_{\text{persp}} \lim_{f \to \infty} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2n}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Por otra parte, muchas veces simplemente queremos un sistema en el que la cámara esté mirando hacia el centro de la ventana frontal del frustrum, esto es que $l = -r$ y $t = -b$. Simplificando así la descripción de la proyección al especificar el ángulo de visión θ. En este caso, el ángulo de visión vertical (FOV).

Una vez más, de forma análoga a la cámra pinhole tenemos la siguiente relación usando trigonometría básica

$$\tan\left(\frac{\theta}{2}\right) = \frac{t}{|n|} \Rightarrow t = tan\left(\frac{\theta}{2}\right)|n|$$

De modo que si consideramos que la ventana fuera cuadrada entonces tendríamos que $r = t$ y $l = b = -t$. Pero como esto no pasa siempre, tendríamos que ajustar a las coordenadas r y l. Ahora bien, supongamos que tenemos una ventana de $n_x \times n_y$ pixeles, con $n_x \neq n_y$. Entonces la proporción entre r y t debería ser la misma que el número de pixeles horizonales y el número de pixeles verticales, esto es $\frac{n_x}{n_y} = \frac{r}{t}$.

Por lo que, las coordenadas r y l deberían ser escaladas por la proporción $\frac{n_x}{n_y}$, también llamada *aspect-ratio*, teniendo

$$\mathbf{M}_{\text{persp}} = \begin{bmatrix} c/a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

donde $a = n_x/n_y$ y $c = 1/tan(\frac{\theta}{2})$ ya que queremos que el vector termine en $[-1, 1]$.

5.3 Transformación de viewport

La transformación de viewport mapea las coordenadas x y y de las coordenadas NDC a coordenadas de pantalla o ventana. Las coordenadas NDC son escaladas y trasladadas de modo que sean ajustadas al tamaño del área de la pantalla sobre la cual se renderizará la escena.

$$\underbrace{\begin{bmatrix} x_{ndc} \\ y_{ndc} \end{bmatrix}}_{\text{coordenadas NDC}} = \underbrace{\begin{bmatrix} x_{ventana} \\ y_{ventana} \end{bmatrix}}_{\text{coordenadas de ventana}}$$

Dicha área se define dentro de la ventana gráfica que es usada por la aplicación. La coordenada z del volumen canónico es ignorada ya que no afectan su posición en la ventana.

De manera general, para transformar los puntos contenidos dentro de un rectángulo $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ a otro rectángulo $[u_{min}, u_{max}] \times [v_{min}, v_{max}]$, simplemente se traslada el punto (x_{min}, y_{min}) al origen, luego se escala el tamaño del primer rectángulo al tamaño del segundo y finalmente se traslada a el punto (u_{min}, v_{min}). A este tipo de transformación se le conoce como *window transformation*.

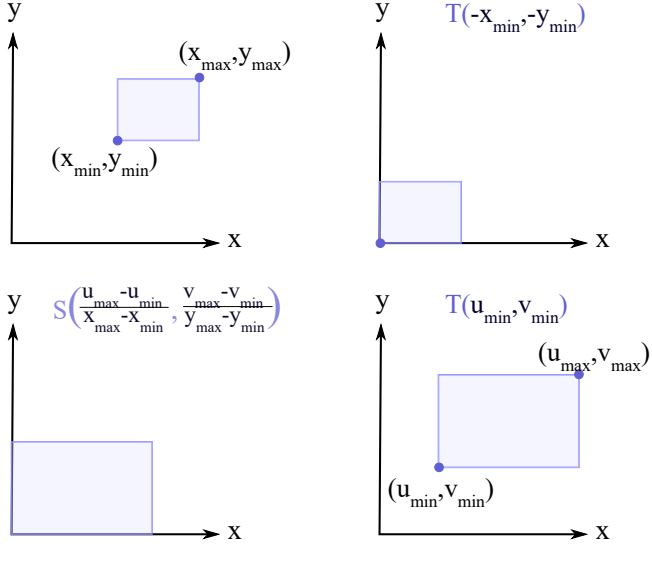


Figura 5.5. Transformación de ventana.

Entonces

$$\mathbf{M}_{\text{window}} = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 1 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & \frac{u_{min} x_{max} - u_{max} x_{min}}{x_{max} - x_{min}} \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & \frac{v_{min} y_{max} - v_{max} y_{min}}{y_{max} - y_{min}} \\ 0 & 0 & 1 \end{bmatrix}$$

Por lo que, si queremos mapear la coordenada $x = -1$ al lado izquierdo de la ventana a renderizar, $x = 1$ al lado derecho de la ventana, $y = -1$ al inferior y $y = 1$ al superior. Es lo mismo que transformar el rectángulo $[-1, 1] \times [-1, 1]$ al rectángulo que conforma el área que queremos renderizar.

Sea $[x, x + w] \times [y, y + h]$ la ventana en la que se va a renderizar, entonces tenemos

$$\mathbf{M}_{\text{vp}} = \begin{bmatrix} \frac{(x+w)-x}{2} & 0 & \frac{2x+w}{2} \\ 0 & \frac{(y+h)-y}{2} & \frac{2y+h}{2} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{w}{2} & 0 & x + \frac{w}{2} \\ 0 & \frac{h}{2} & y + \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Nótese que este tipo de transformación es similar a la que se usa para las transformaciones de proyección con la cual se ajusta el volumen de visión al volumen canónico de vista, solo que se define en 3D, transformando una caja $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ a la caja $[u_{min}, u_{max}] \times [v_{min}, v_{max}] \times [w_{min}, w_{max}]$, esto es

$$\mathbf{M}_{\text{window}} = \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 & \frac{u_{min} x_{max} - u_{max} x_{min}}{x_{max} - x_{min}} \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 & \frac{v_{min} y_{max} - v_{max} y_{min}}{y_{max} - y_{min}} \\ 0 & 0 & \frac{w_{max} - w_{min}}{z_{max} - z_{min}} & \frac{w_{min} z_{max} - w_{max} z_{min}}{z_{max} - z_{min}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.4 Referencias bibliográficas

[95] Shirley, Peter, et al. ***Fundamentals of Computer Graphics.*** A K Press. 3ª Edición. 2009.

[96] Lengye, Eric. ***Mathematics for 3D Game Programming and Computer Graphics.*** Course Technology, a part of Cengage Learning. 3ª Edición. 2012.

[97] Akenine-Möller, Tomas, et al. ***Real-Time Rendering.*** A K Peters/CRC Press. 4ª Edición. 2018.

[98] Tumblin, Jack. 2016. ***EECS 351-1 : Introduction to Computer Graphics: Building the Virtual Camera ver. 1.5***. Northwestern - Computer Science. https://canvas.northwestern.edu/files/1973650/download?download_frd=1

[99] Ho Ahn, Song. 2008. ***OpenGL Transformation***. Songho.ca. http://www.songho.ca/opengl/gl_transform.html

[100] Ho Ahn, Song. 2016. ***OpenGL Projection Matrix***. Songho.ca. https://www.songho.ca/opengl/gl_projectionmatrix.html

[101] Ho Ahn, Song. 2016. ***OpenGL Camera***. Songho.ca. https://www.songho.ca/opengl/gl_camera.html

CAPÍTULO VI

Modelos de iluminación y sombreado

> *"In computer graphics, a shading function is defined as a function which yields the intensity value of each point on the body of the object from the characteristics of the light source, the object, and the position of the observer."* - Bui Tuong Phong (1975).

La interacción entre la luz y una superficie es una proceso físico bastante complejo. La luz está compuesta por *fotones*, los cuales son un tipo de partícula elemental, que pueden ser pensados como paquetes muy pequeños de energía que se mueven a una velocidad constante y no poseen masa.

En esta representación, la luz puede pensarse como un flujo de muchos paquetes de energía, los cuales son propagados de un lugar a otro en forma de rayo. Dichos rayos solo pueden ser redirigidos por su interacción con la materia, y cada vez que éstos chocan contra una superficie un porcentaje de fotones puede ser absorbido, reflejado o transmitido dependiendo de las propiedades reflectivas de la superficie y de la luz misma. En la **Figura 6.1** se pueden apreciar dichos efectos. De modo que este proceso continua repitiéndose, es decir, cierta cantidad de energía sigue viajando en el entorno, hasta que toda la energía es transferida, ver **Figura 6.2**.

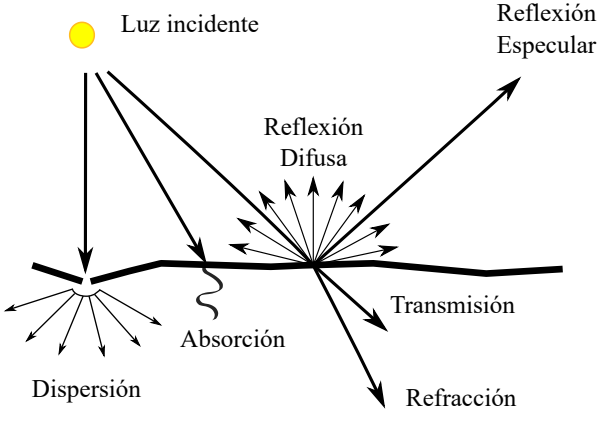


Figura 6.1. Interacciones básicas entre la luz y una superficie.

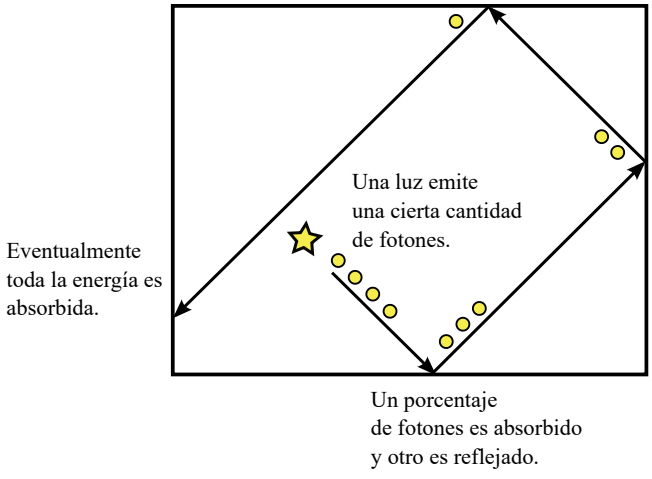


Figura 6.2. Reflexiones de luz sobre un conjunto de superficies.

En el mundo real, las superficies reciben luz de otras superficies incluso si éstas no producen luz o no son iluminadas directamente. Por lo que podemos clasificar a las fuentes de luz en dos tipos: como emisor de luz o fuente de luz primaria, las cuales producen la luz que emiten, o bien, un reflector de luz o fuente secundaria, es decir, aquellas que reflejan la luz que producen otras fuentes.

Por ejemplo, pensemos en la iluminación de un cuarto durante el día, donde la luz del sol solo entra por una ventana. Si bien, algunas partes del cuarto serán iluminadas directamente por la luz del sol, hay otras partes como el techo, que no. Sin embargo, éstas partes se ven iluminadas gracias a las reflexiones de luz generadas por aquellas que sí son iluminadas, como el piso y las paredes, y que eventualmente alcanzan a estas partes no iluminadas.

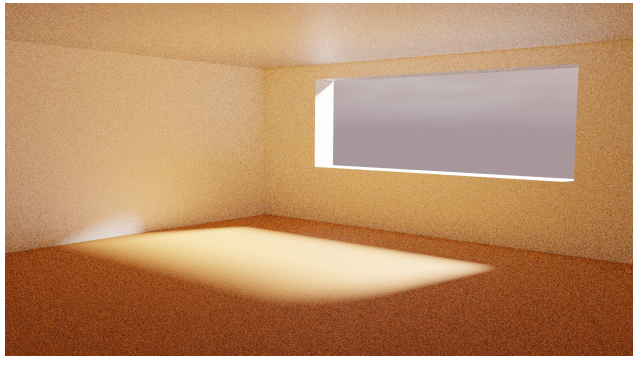


Figura 6.3. Iluminación global. Cuarto con una ventana. Imagen generada con blender.

Esta dispersión de luz recursiva entre las superficies explica los efectos sutíles del sombreado, tales como las sombras producidas entre objetos adyacentes, la luz indirecta que es la cantidad de luz recibida a través de las reflexiones de luz de objetos cercanos, o bien, el *color bleeding* que es un efecto particular de la luz indirecta, el cual describe el modo en que el color de un objeto es influenciado por el color de otros objetos que lo rodean. En la **Figura 6.3** puedes observar estos efectos. A este tipo de iluminación se le conoce como *Iluminación global*.

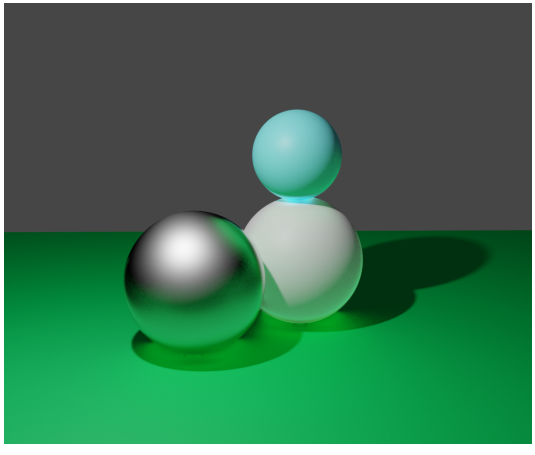


Figura 6.4. Efectos de iluminación global: *color bleeding* y sombreado. Imagen generada con blender.

Y ya que la luz viaja demasiado rápido (cerca de $3 \times 10^8 \text{m/sec}$) no importa el tamaño de la escena, la luz tardará un tiempo insignificante en propagarse por todo el entorno.

Sin embargo, reproducir la iluminación global en una escena virtual resulta ser una tarea bastante compleja y computacionalmente costosa, pues requiere demasiado tiempo de cómputo para calcular todas las posibles interacciones de luz entre los objetos de una escena.

Por lo que para facilitar los cálculos nos enfocaremos en modelos de *Iluminación local*, en donde evaluaremos únicamente las contribuciones de las fuentes de luz primarias sobre un punto de la superficie, con las cuales determinaremos el color del punto, considerando siempre que la fuente de luz y la superficie son visibles. Por lo que si una superficie debería generar una sombra sobre otra, ésta no se vería, teniendo un resultado menos realista. Por otro lado, el color que se le asignará al punto dependerá tanto de las propiedades de las fuentes de luz que iluminen a la superficie, como de las pro-

piedades del material que compone la superficie misma, y de ser el caso de la posición del observador. Esto con el fin de entender los conceptos básicos de iluminación tales como modelos de iluminación y tipos de fuentes de luz.

Nos referiremos a un punto en la superficie como una pequeña area en alguna cara que componga al objeto que queremos colorear, siendo más específicos, a un fragmento. Por lo que la técnica de sombreado que usaremos en la implementación de los interactivos a lo largo del capítulo será la técnica de *Phong shading*, siendo una de las más populares.

Ahora bien, como vimos anteriormente, en el Sistema de visión humano, la luz también puede comportarse como una onda y solo podemos percibir aquella que esté dentro del espectro visible. La luz visible puede estar compuesta por una única longitud de onda (monocromática) o varias longitudes de onda (policromática). De modo que cuando un objeto es iluminado por una fuente de luz, el color que percibimos que tiene es el resultado de la luz que está reflejando o longitudes de onda que no absorbió.

Por ejemplo, si una luz blanca (la cual contiene todos los colores o longitudes de onda del espectro visible) choca contra un objeto y éste se ve de un color azul turquesa, quiere decir que ha absorbido todos los colores, menos los azules y verdes (en su mayoría), que son los que componen al color que refleja.

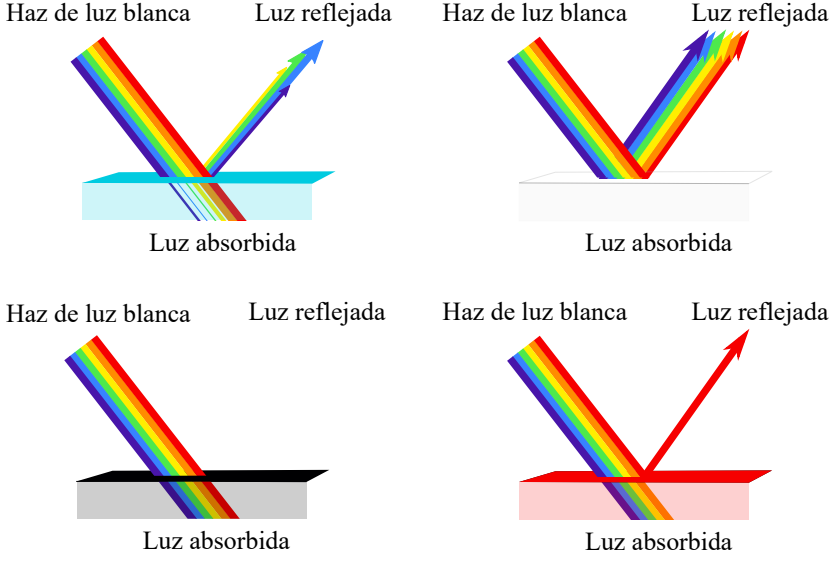


Figura 6.5. Reflexión y absorción de la luz en una superficie.

Entonces para crear una imagen realista de nuestra escena virtual en 3D necesitamos determinar las longitudes de onda de luz que el objeto refleja.

Afortunadamente, con el modelo RGB podemos aproximarnos a cada color del espectro visible por medio de una combinación lineal de tres colores primarios (R, G, B). Los modelos de iluminación que presentaremos utilizarán este modelo de color, evaluando cada componente en un rango de 0 a 1, con los cuales representaremos la composición espectral de la luz, es decir, el color que percibimos, así como su intensidad. Esto definiendo la intensidad o cantidad de luz que reflejan de cada canal R, G y B.

Entonces, podemos modelar esta interacción de la siguiente manera. Consideremos que tenemos una fuente de luz blanca que ilumina a un objeto, y de acuerdo a sus propiedades éste refleja un color verde bosque, entonces el color que veríamos estaría dado por

$$\underbrace{(1.0, 1.0, 1.0)}_{\text{luz blanca}}\underbrace{(0.121, 0.498, 0.121)}_{\text{color del objeto}} = \underbrace{(0.121, 0.498, 0.121)}_{\text{color reflejado}}$$

definiendo así al color reflejado como la cantidad o intensidad de luz que cada componente refleja de la luz que recibe.

Por otro lado, si utilizáramos luz monocromática de color rojo entonces solo el componente rojo de la luz sería reflejado, por lo que percibiríamos a nuestro objeto de un color casi negro, es decir, sin color, a excepción de un ligero tono rojo

$$\underbrace{(1.0, 0.0, 0.0)}_{\text{luz roja}}\underbrace{(0.121, 0.498, 0.121)}_{\text{color del objeto}} = \underbrace{(0.121, 0.0, 0.0)}_{\text{color reflejado}}$$

Sin embargo, si repetimos este mismo cálculo para todos los puntos del objeto tendremos como resultado un objeto con un color uniforme y plano, como se puede apreciar en la **Figura 6.6**, por lo que necesitarémos determinar el color que deberá tener el objeto en cada uno de sus puntos.

Durante este capítulo se describirán los cálculos usados para iluminar y sombrear una superficie utilizando iluminación local, obteniendo así una aproximación aceptable de cómo debería lucir un objeto en el mundo real.



Figura 6.6. Se invita al lector a modificar el color de la fuente de luz y el color que refleja el objeto.

6.1 Reflexión ambiental

La reflexión o luz ambiental de una escena es la luz de baja intensidad que surge de los numerosos reflejos de luz de todas las superficies cercanas en el entorno, también conocida como *luz indirecta*.

La reflexión ambiental nos proporciona una aproximación de la luminosidad de la escena en general, reemplazando la compleja tarea de calcular la cantidad de luz que recibe una superficie dadas las reflexiones de otros objetos dentro de ésta, ya que a pesar de tener una escena muy poco iluminada, los objetos casi nunca son completamente negros. Y dependiendo del material del objeto éste podrá absorber y reflejar mayor o menor cantidad de luz ambiental.

Para cada punto P en la superficie, la intensidad de luz ambiental reflejada puede ser modelada de la siguiente forma. Sea $\mathbf{L}_A$ la cantidad de luz producida por la luz incidente, y k_A el coeficiente de reflexión ambiental característico del material, con $0 \leq k_A \leq 1$, entonces

$$\begin{aligned} \mathbf{I}_P &= k_A \mathbf{L}_A \\ &= k_A(L_{A,R}, L_{A,G}, L_{A,B}) \end{aligned}$$

Agregando una cierta cantidad de luz que parece provenir de todas las direcciones con la misma intensidad, iluminando cada parte de un objeto de manera uniforme. Sin embargo, aunque puede ser un efecto valioso debe usarse cuidadosamente, ya que al aumentar la luminosidad de todo lo que hay en la escena, puede llegar a desvanecer algunos detalles como las sombras.

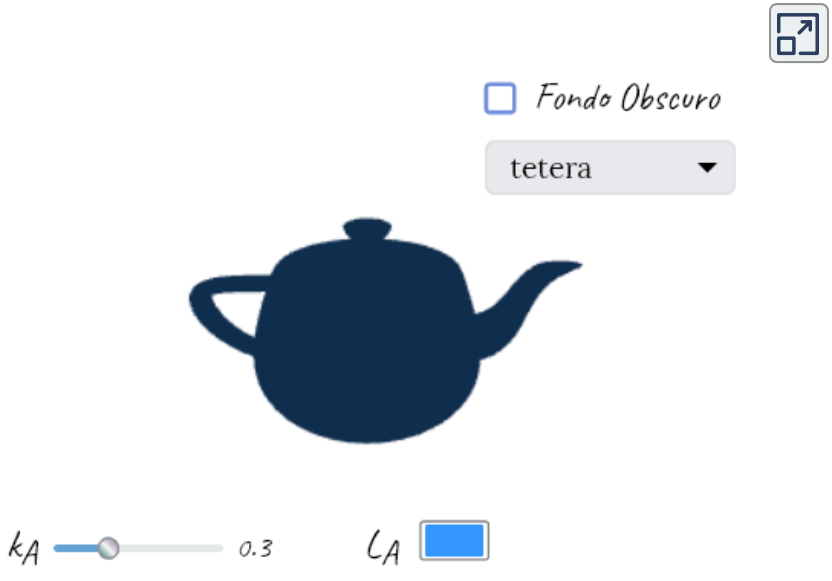


Figura 6.7. Reflexión ambiental. Se invita a modificar el valor de k_A y el color de L_A. Puedes cambiar el modelo y mover la cámara con el ratón o dedo.

6.2 Reflexión difusa (superficies de Lambert)

Las superficies puramente difusas son aquellas que reflejan la luz uniformemente en todas las direcciones, por lo que tendrá la misma apariencia sin importar la posición del observador.

La reflexión difusa es característica de superficies de materiales rugosos, mates y sin brillo. Por ejemplo, si miras fijamente a un punto determinado de una hoja de papel y te mueves, éste seguirá viéndose del mismo color. Pues si ampliáramos una sección de una superficie difusa veríamos algo aparecido a la **Figura 6.8**, donde los rayos de la luz incidente son dispersados por igual en direcciones aleatorias.

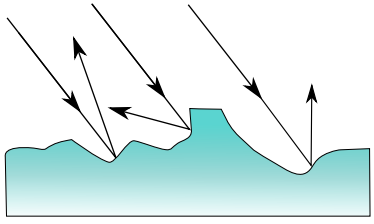


Figura 6.8. Superficie difusa (a nivel microscópico). Reflexión de la luz en direcciones aleatorias.

Este tipo de materiales suelen ser modelados usando la *Ley del Coseno de Lambert*, por esta razón es que también son llamadas como superficies de Lambert. La ley del coseno nos dice que la cantidad de luz reflejada en una pequeña área o punto de una superficie depende del ángulo formado entre el vector normal **N** en el punto y la dirección de la luz incidente **L**, es decir, el vector que va del punto a la fuente de luz.

Consideremos una pequeña área A de un superficie plana, la cual es iluminada por un haz de luz, véase la **Figura 6.9**. De modo que cuando el haz de luz es perpendicular a A todos los fotones contenidos en el haz chocan con A, permitiendo reflejar la mayor cantidad de fotones. Y como podemos notar en el interactivo, al incrementar el ángulo de inclinación θ formado entre el haz de luz y la normal de A la cantidad de fotones que cae en A se reduce, pues la sección transversal que el haz de luz cubre se vuelve más grande, por lo que la luz es esparcida y una menor cantidad de fonotes o energía ilumina a A. Por otro lado, si el haz de luz es paralelo a la superficie, A no es iluminada, pues no recibe nada de energía.

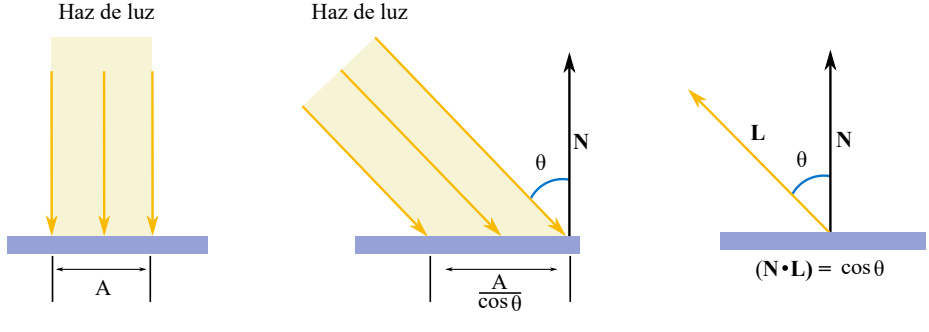


Figura 6.9. La figura de la izquierda muestra el área A de la superficie que impacta el haz de luz, la figura del centro se observa como el haz impacta la superficie con cierto ángulo y la de la derecha muestra el cálculo de dicho ángulo.

De estas observaciones podemos concluir que A recibe la mayor cantidad de energía cuando el haz de luz es perpendicular a la superficie, en otras palabras, cuando $\mathbf{L}$ es paralelo a $\mathbf{N}$, y esta cantidad va disminuyendo conforme el ángulo θ incrementa, volviéndose más oscura cada vez. Finalmente cuando $\mathbf{L}$ es perpendicular a $\mathbf{N}$ el área A no es iluminada, por lo que no refleja energía. Entonces podemos decir que la cantidad de luz que se refleja por el área es inversamente proporcional a el coseno de θ, esto es $A/\cos\theta$.

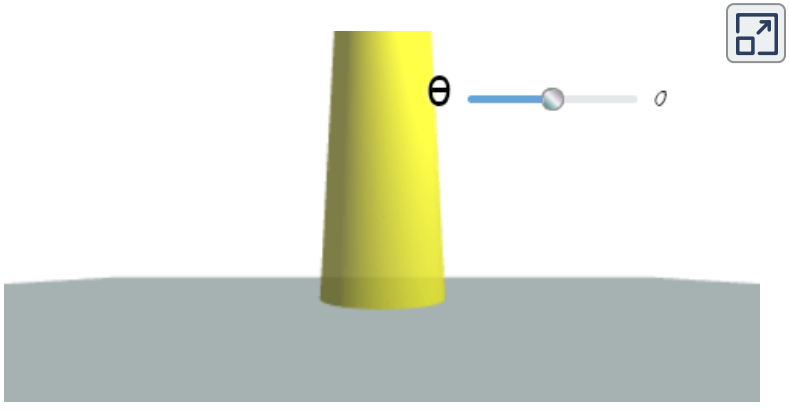


Figura 6.10. Iluminación de un área de una superficie por un haz de luz. Se invita al lector a rotar el haz de luz representado por el cilindro amarillo y observar el área de la superficie que llega a impactar. Cuando el haz de luz es perpendicular a la superficie, una área más pequeña recibe todos los fotones del haz de luz. Conforme rotamos el haz notamos que el área que cubre es mayor, distribuyendo la cantidad de fotones por la superficie, hasta ser paralelo y no transmitir energía.

El valor $\cos\theta$ puede obtenerse con el producto punto de los vectores normalizados $\mathbf{N}$ y $\mathbf{L}$. Y como el producto punto puede ser negativo cuando la superficie apunta en dirección contraria a la luz, es decir, la fuente de luz se encuentra atrás de la cara frontal de la superficie, como se muestra en la **Figura 6.11**, el valor del producto punto se suele limitar a $[0, 1]$, ya que de otro modo tendríamos "luz negativa".

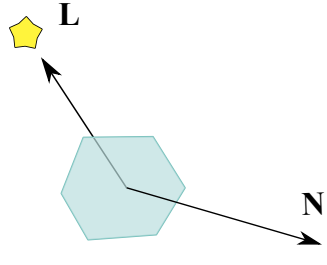


Figura 6.11. Superficie apuntando en dirección contraria a la fuente luz.

Este modelo es conocido como el Modelo de Lambert debido a su dependencia a la ley de Lambert, el cual se define a continuación.

Para cada punto P en la superficie, podemos calcular la intensidad de luz difusa reflejada en P como

$$\begin{aligned}\mathbf{I}_P &= k_D \mathbf{L}_D \max(\cos\theta, 0) \\ &= k_D \mathbf{L}_D \max((\mathbf{N} \cdot \mathbf{L}), 0)\end{aligned}$$

con $\mathbf{N}$ y $\mathbf{L}$ normalizados. Donde $\mathbf{L}_D$ es la cantidad de luz producida por la luz incidente, y k_D el coeficiente de reflexión difusa de la superficie que nos indica qué tan difusa es, siendo $0 \leq k_D \leq 1$.

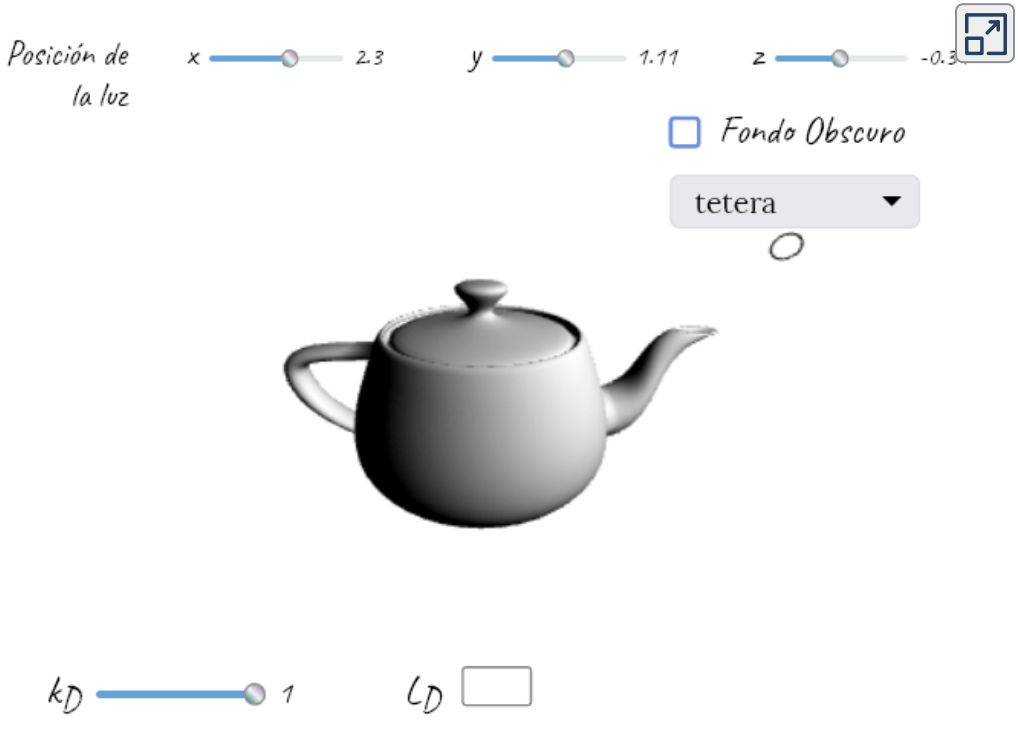


Figura 6.12. Reflexión difusa. Se invita a modificar el valor de k_d y el color de L_S, así como los valores de la posición de la fuente de luz. Puedes cambiar el modelo y mover la cámara con el ratón o dedo.

6.3 Reflexión especular

La reflexión especular produce reflejos brillantes, se presenta en superficies brillantes, pulidas o satinadas, como metales y vidrios. La reflexión especular a diferencia de la reflexión difusa y ambiental depende tanto de la dirección de la luz incidente como de la posición del observador.

A nivel microscópico una superficie especular es muy lisa, por ello los rayos de luz rebotan como un espejo ideal, como se aprecia en la **Figura 6.13**. Y mientras más lisa o menos irregularidades tenga la superficie, más se asemeja a un espejo y se le conoce como superficie especular perfecta.

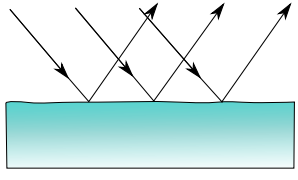


Figura 6.13. Superficie especular (a nivel microscópico).

La **Figura 6.14** muestra una reflexión especular perfecta (izquierda), en este caso un material puramente especular refleja toda la luz incidente con exactamente el mismo ángulo de incidencia, a este vector espejo se le conoce también como *vector de reflexión perfecta*. Por otro lado, cuando se tiene una superficie especular no ideal (derecha), una parte de la energía recibida se refleja a lo largo del vector de reflexión, y otra cantidad se refleja de manera difusa al rededor del vector espejo, el cual tiene como efecto el difuminar la luz reflejada sobre la superficie, actuando como un espejo roto.

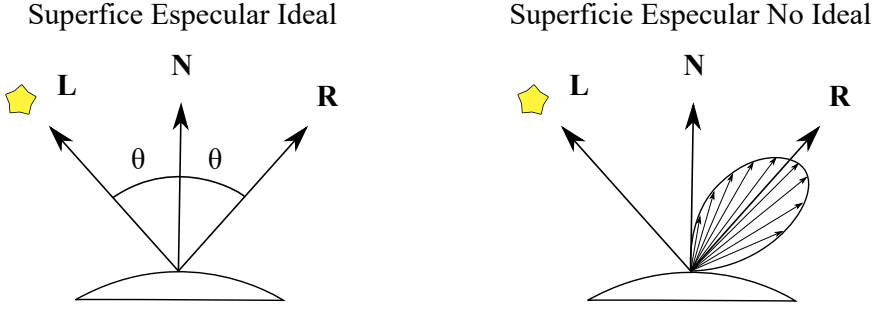


Figura 6.14. Reflexión especular ideal o perfecta (izquierda). Reflexión especular no ideal o espejo rugoso (derecha).

Sea $\mathbf{N}$ el vector normal y $\mathbf{L}$ la dirección de la luz incidente, y ambos normalizados. Podemos calcular el vector de reflexión $\mathbf{R}$ como se muestra en la **Figura 6.15**

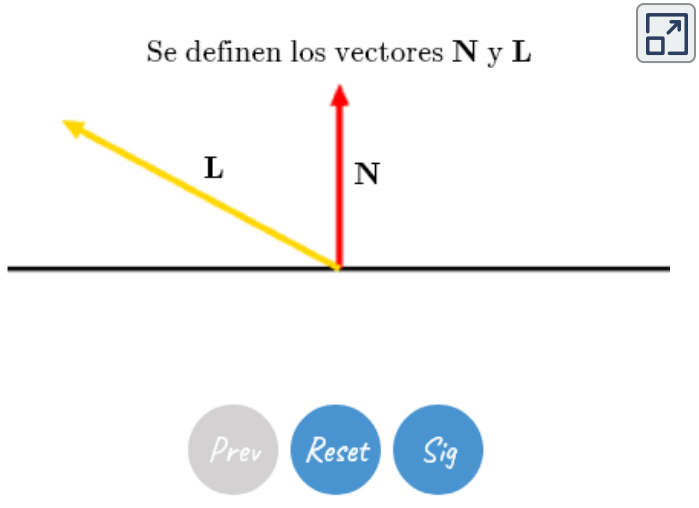


Figura 6.15. Cálculo del vector espejo. El vector $\mathbf{L}$ puede ser modificado arrastrando la punta de la flecha en el primer paso. Da clic en el botón *Sig* o *Prev* para ver el paso siguiente o previo.

Entonces podemos expresar al vector $\mathbf{R}$ como

$$\mathbf{R} = 2\mathbf{N}(\mathbf{N} \cdot \mathbf{L}) - \mathbf{L}$$

Y si asumimos que el observador está viendo hacia la superficie en la misma dirección que el vector espejo $\mathbf{R}$, ésta podrá reflejar un brillo más tenue o más fuerte dependiendo del material de la superficie. Si tenemos una superficie no ideal entonces es claro que se reflejá un brillo más tenue ya que algunos de los rayos serán dispersados en direcciones aleatorias, mientras que en una superficie perfecta, todos los rayos serán reflejados hacia el ojo u observador.

Ahora bien, si el observador estuviera posicionado diferente, es decir, el vector de vista o *view direction* $\mathbf{V}$ que va del punto en la superficie a la cámara es distinto al vector $\mathbf{R}$, ver **Figura 6.16**. La cantidad de energía que se refleja, o mejor dicho, que nuestro observador ve en un punto, decrementa a medida que el ángulo ϕ formado entre $\mathbf{R}$ y $\mathbf{V}$ se incrementa. Esto es, ya que a medida en que ϕ se incrementa, el número de rayos que son reflejados en la misma dirección de $\mathbf{V}$ disminuye.

Para calcular el vector $\mathbf{V}$ que va de la posición del fragmento a la cámara basta restar la posición de la cámara con la posición del fragmento. Y si realizamos los cálculos desde el espacio de la cámara, como discutimos previamente en el Capítulo 5, ésta se encontraría posicionada en el origen. Por lo que considerando a P como el punto en la superficie, tendríamos que $\mathbf{V} = -P$.

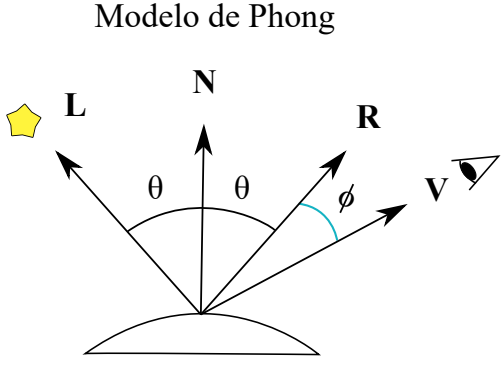


Figura 6.16. Reflexión especular del Modelo de Phong. La intensidad de la reflexión especular está dada en función del ángulo formado entre el vector $\mathbf{V}$ y $\mathbf{R}$.

Phong observó que este brillo podía ser simulado de la siguiente forma. Tomando en consideración que los vectores $\mathbf{V}$ y $\mathbf{R}$ son vectores unitarios, así como las características de las propiedades de la superficie y de la luz. Para cada punto P en el objeto podemos calcular la intensidad de la luz especular recibida por el ojo como

$$\begin{aligned}\mathbf{I}_P &= k_S \mathbf{L}_S \max(\cos(\phi), 0)^\alpha \\ &= k_S \mathbf{L}_S \max((\mathbf{R} \cdot \mathbf{V}), 0)^\alpha\end{aligned}$$

siendo $\mathbf{L}_S$ la cantidad de luz producida por la luz incidente, k_S la fracción de la luz especular reflejada del material, con $0 \leq k_S \leq 1$, con $\mathbf{R}$ y $\mathbf{V}$ normalizados. El valor del producto punto es utilizado para simular la reflexión especular y el exponente especular α es un número positivo que controla el tamaño y definición del brillo, el cual es ajustado empíricamente de acuerdo a las propiedades reflexivas del material del objeto.

A este modelo se le conoce como *Modelo especular de Phong*, ya que fue el que desarrolló para su modelo de iluminación, del cual hablaremos un poco más adelante.

Como se puede observar en la **Figura 6.17**, a medida que el componente especular se va haciendo más grande la curva se va estrechando al rededor del eje y. De modo que un valor pequeño para α produce un brillo mate que se desvanece a una distancia relativamente larga, mientras que un valor grande genera un brillo más definido y pequeño que se desvanece tan rápido como los vectores $\mathbf{R}$ y $\mathbf{V}$ divergen.

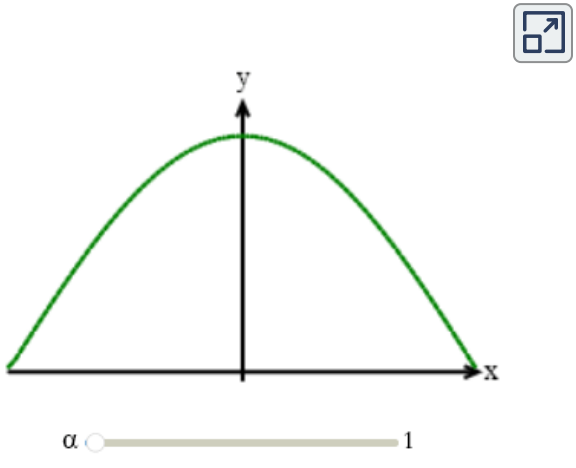


Figura 6.17. Gráfica de la función $\cos\phi^{\alpha}$ del brillo especular de Phong.

Aunado a ello, vale la pena verificar sí $((\mathbf{N} \cdot \mathbf{L}) > 0)$ antes de calcular la reflexión especular, ya que de manera similar a la reflexión difusa, la fuente de luz estaría detrás de la superficie y no debería presentar un brillo especular.

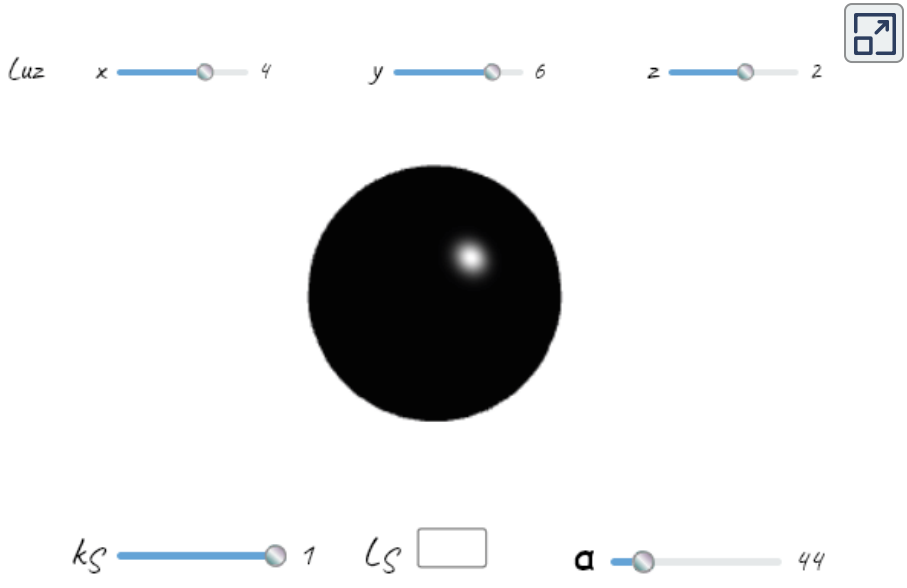


Figura 6.18. Reflexión especular. Se invita al lector a modificar el valor del exponente especular α, con lo cual podemos simular varias cantidades de luz dispersada al rededor del vector de vista. Así como también observar la dependencia de la posición del observador moviendo la posición de la cámara con el ratón o dedo.

6.4 Modelo de iluminación de Phong

El Modelo de iluminación de Phong es uno de los modelos de iluminación local más utilizados el cual fue desarrollado de forma empírica por Bui Tuong Phong en 1975.

En el mundo real la mayoría de las superficies no suelen ser puramente difusas o especulares, más bien, suelen estar compuestas por ambas propiedades. Por lo que, la luz recibida por el observador es producida una parte por la reflexión difusa y otra parte por la reflexión especular de la luz incidente. Tomando en consideración esto, Phong describió a la luz reflejada por un punto P en la superficie de un objeto como

$$\mathbf{I}_P = \underbrace{k_A \mathbf{L}_\mathrm{A}}_{\text{Reflexión ambiental}} + \underbrace{k_D \mathbf{L}_D \max((\mathbf{N}\cdot\mathbf{L}),0)}_{\text{Reflexión difusa}} + \underbrace{k_S \mathbf{L}_S \max((\mathbf{R}\cdot\mathbf{V}),0)^\alpha}_{\text{Reflexión especular}}$$

donde se incorpora cada uno de los modelos de reflexión que hemos visto, permitiendo modelar cada una de las propiedades reflexivas del material.

Y en el caso de tener múltiples fuentes de luz, la reflexión de cada una de estas luces en P es acumulada para así obtener la cantidad total de luz reflejada. Esto es

$$\mathbf{I}_P = k_A L_A + \sum_{i=0}^{n} (k_D \max((\mathbf{N} \cdot \mathbf{L}_{D_i}), 0)) + k_S \max((\mathbf{R}_i \cdot \mathbf{V}), 0)^{\alpha}$$

siendo n el número de luces.

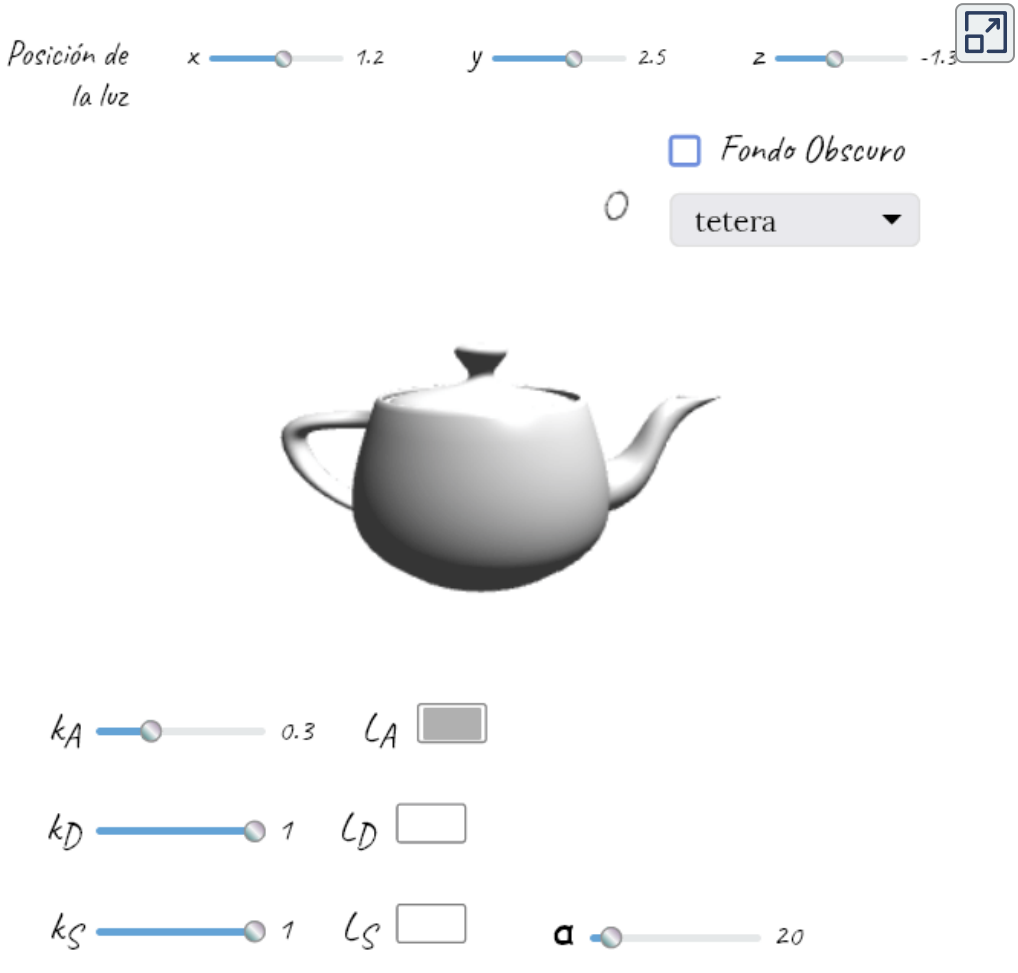


Figura 6.19. Modelo de iluminación de Phong. Se invita a modificar cada uno de los componentes del modelo, así como cambiar la posición de la luz y el modelo a utilizar. Puedes mover la posición de la cámara con el ratón o dedo.

6.5 Modelo de iluminación Blinn-Phong

El Modelo de iluminación de Blinn-Phong es una variación del modelo de Phong propuesta por James Blinn. En esta adaptación se buscó simplificar el cálculo de la reflexión especular, ya que necesitaba calcular el vector de reflexión $\mathbf{R}$ para cada punto de la superficie. En su lugar, se propone calcular el vector intermedio o *half vector* $\mathbf{H}$ para determinar la intensidad del brillo especular, el cual es el vector que se encuentra entre el vector de vista $\mathbf{V}$ y el de la luz incidente $\mathbf{L}$, como se muestra en la **Figura 6.20**.

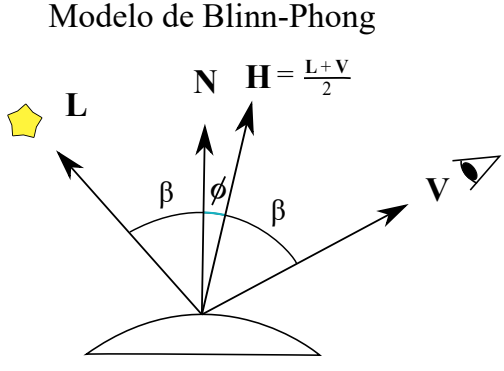


Figura 6.20. Modelo de ilumación de Blinn-Phong..

Entonces en el modelo de Phong la reflexión especular ahora se calcularía en función del ángulo formado entre el vector intermedio y la normal del punto como

$$\begin{aligned}\mathbf{I}_P &= k_S \mathbf{L}_S \max(\cos\phi), 0)^\alpha \\ &= k_S \mathbf{L}_S \max((\mathbf{N} \cdot \mathbf{H}), 0)^\alpha\end{aligned}$$

$$\text{donde } \mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{2}$$

Nótese que todos los vectores mencionados están normalizados.

De manera similar, el ángulo ϕ nos da una aproximación de qué tan cerca se encuentra el vector de vista del vector de reflexión o espejo. De modo que el brillo especular es más intenso a media que $\mathbf{H}$ se acerca al vector normal $\mathbf{N}$.

Sin embargo, como se puede observar en la **Figura 6.20**, el ángulo ϕ suele ser más pequeño que el ángulo formado entre $\mathbf{R}$ y $\mathbf{V}$ del modelo original, produciendo diferentes resultado para α. Esta diferencia la podemos observar en la **Figura 6.21**.

Este modelo nos permite optimizar los cálculos, y es por ello que se ha vuelto el modelo más utilizado en gráficos 3D de tiempo real.

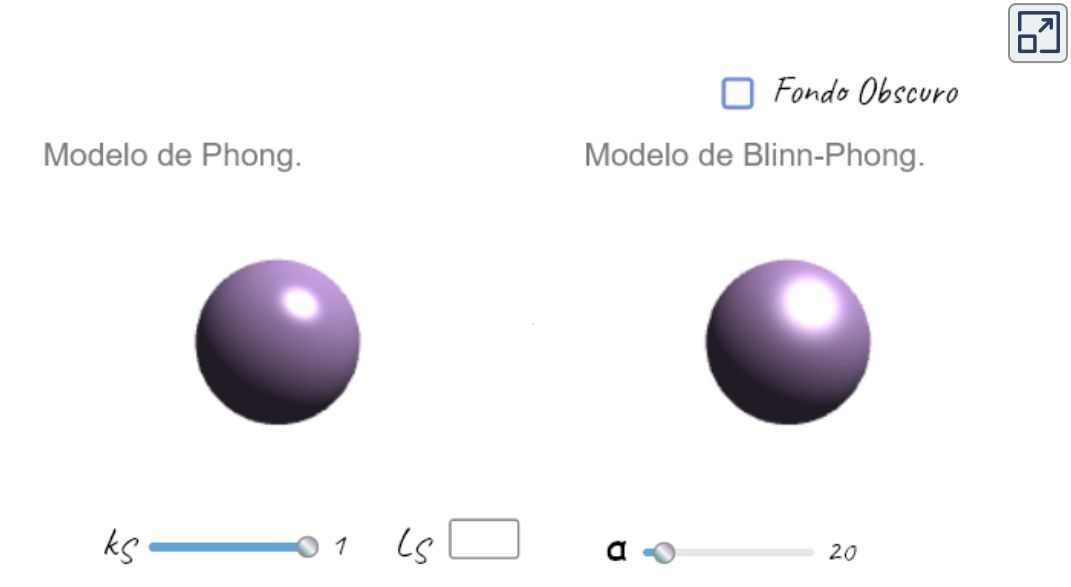


Figura 6.21. Comparación entre el Modelo de iluminación de Phong y Blinn-Phong. Se invita al lector a modificar el valor del exponente especular y observar la diferencia a pesar de que todas las características del objeto y la luz son las mismas.

6.6 Materiales

Como ya hemos mencionado la intensidad de la luz reflejada por una superficie depende tanto de las propiedades de la luz incidente como de las propiedades reflexivas del objeto.

Veamos algunos ejemplos. En el mundo real, si un material es dieléctrico, es decir, no es conductor eléctrico, y presenta un brillo especular como la madera pulida, la porcelana, el plástico, etc, además de reflejar el color que percibimos del objeto, se refleja un brillo especular del mismo color que emite la luz incidente. Por otro lado, si se trata de un material metálico, el brillo especular

reflejado será del color del metal mismo, por ejemplo el oro refleja un brillo especular de color amarillo.

Utilizando el modelo de iluminación de Phong podemos modelar estas propiedades. Podemos especificar las propiedades reflexivas del material con los coeficientes de reflexión k_A, k_D y k_S, los cuales representaremos como tripletas (R, G, B) para definir así las intensidades de cada uno de los canales de color que refleja cada componente, y del mismo modo para las propiedades de la luz incidente con L_A, L, D y L_S. Entonces el resultado del color reflejado en el punto sería representando como

$$\mathbf{I}_P = \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} k_{A_R} L_{A_R} \\ k_{A_G} L_{A_G} \\ k_{A_B} L_{A_B} \end{bmatrix} + \begin{bmatrix} k_{D_R} L_{D_R} \\ k_{D_G} L_{D_G} \\ k_{D_B} L_{D_B} \end{bmatrix} \max((\mathbf{N} \cdot \mathbf{L}), 0) +$$

$$\begin{bmatrix} k_{S_R} L_{S_R} \\ k_{S_G} L_{S_G} \\ k_{S_B} L_{S_B} \end{bmatrix} \max((\mathbf{R} \cdot \mathbf{V}), 0)^{\alpha}$$

Podemos definir entonces las características del material de una superficie como sigue

```
Material {
  Vector3 ambiental;      //k_A
  Vector3 difuso;         //k_D
  Vector3 especular;      //k_S
  float brillo_especular;
}
```

donde `brillo_especular` hace referencia al exponente especular α.

Análogamente podemos empezar a definir las propiedades de una luz como sigue

```
Luz {
  Vector3 posición;
  Vector3 ambiental;  //L_A
  Vector3 difuso;     //L_D
  Vector3 especular;  //L_S
}
```

obteniendo así la dirección de la luz incidente **L** como el vector que va de la posición de la fuente de luz a un punto de la superficie. En el siguiente subtema veremos otras formas de modelar a una fuente de luz.

Por otro lado, la intensidad del componente ambiental de la luz L_A suele ser un color bajo o tenue ya que no queremos que sea el que predomine en el objeto, ni perder detalles. La intensidad difusa L_A suele ser exactamente el color de la luz o un color brillante como el blanco para reflejar así el color base que percibimos del objeto. Por último el color especular L_S suele ser el blanco en el caso de materiales no metálicos como mencionamos al principio o del color del metal.

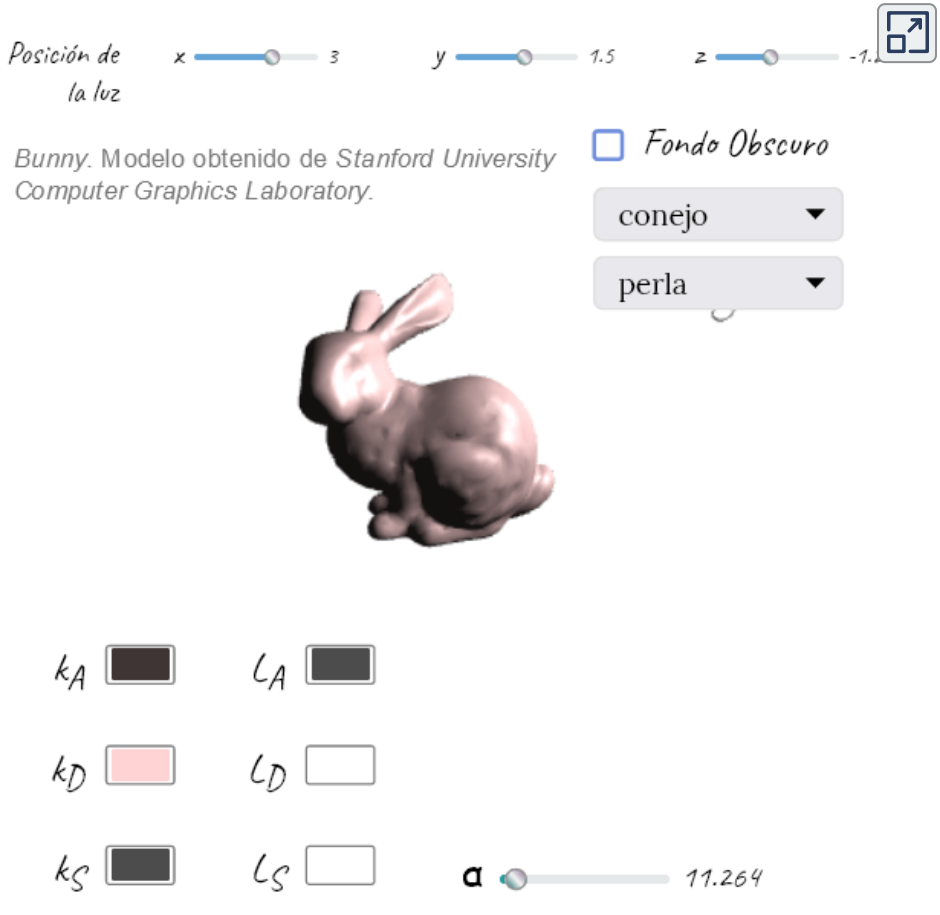


Figura 6.22. Puedes cambiar los modelos y materiales a aplicar, así como mover la posición de la cámara con el ratón o dedo. Da clic en "personalizar" para modificar los valores de reflexión del material.

En esta lista encontrarás la descripción de distintos materiales reales para poder simularlos.

6.7 Fuentes de luz

Recordemos que por fuente de luz nos referimos a un punto en el espacio 3D que produce o emite energía. Hasta ahora hemos modelado a las fuentes de luz únicamente por su posición, a partir de la cual se emiten rayos de luz en todas las direcciones con una misma intensidad, lo cual no es muy realista.

En el mundo real existe una variedad de fuentes de luz, con diferentes formas, tamaños y colores. Por ejemplo una lámpara de la calle, una vela, el sol o luces de neón. A continuación veremos cómo modelar algunas de las fuentes de luz más utilizadas.

6.7.1 Luz puntual

Una fuente de luz puntual es un punto en el espacio que emite energía en todas las direcciones dentro de un campo esférico y su intensidad se debilita con la distancia. Un ejemplo de este tipo de fuente de luz es un foco o una vela.

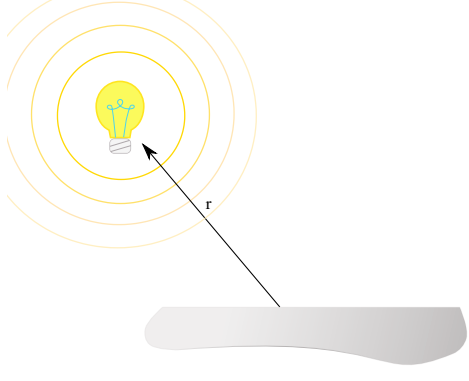


Figura 6.23. Luz puntual o de posición.

La intensidad de la luz recibida en un punto de una superficie disminuye de acuerdo a la Ley del cuadrado inverso, esto es, su intensidad es inversamente proporcional al cuadrado de la distancia d que hay entre la posición de la luz y el punto. Aunque esto es matemáticamente correcto en el mundo real, los resultados no suelen ser muy naturales ya que la intensidad se desvanece rápidamente a medida que la distancia se incrementa.

Por ello suele ser reemplazado por el siguiente factor

$$\text{atenuación} = \frac{1}{ad^2 + bd + c}$$

donde los coeficientes a, b y c son constantes características de la fuente de luz que nos permiten suavizar y controlar mejor la iluminación como se puede observar en la **Figura 6.24**. La constante c suele ser 1 para mantener el denominador mayor a 1. Mientras que el coeficiente b determina el comportamiento lineal de la curva de la atenuación, y a el comportamiento cuadrático.

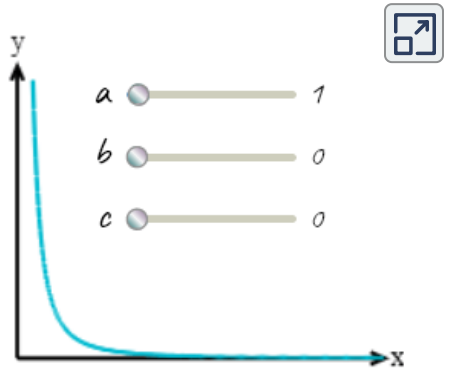


Figura 6.24. Modifique los coeficientes de atenuación y observe cómo afecta la curva.

Por lo tanto, agregando este factor de atenuación al modelo de Phong tenemos

$$\mathbf{I}_P = k_A \mathbf{L}_\mathrm{A} + \frac{1}{ad^2 + bd + c}(k_D \mathbf{L}_D \max((\mathbf{N} \cdot \mathbf{L}), 0) + k_S \mathbf{L}_S \max((\mathbf{R} \cdot \mathbf{V}), 0)^\alpha)$$

Entonces la luz puntual puede ser modelada como sigue

```
Luz_puntual {
  Vector3 posición;
  Vector3 ambiental;  //L_A
  Vector3 difuso;     //L_D
  Vector3 especular;  //L_S

  //Coeficientes de atenuación
  float cuadrático;  //a
  float lineal;      //b
  float constante;   //c
}
```

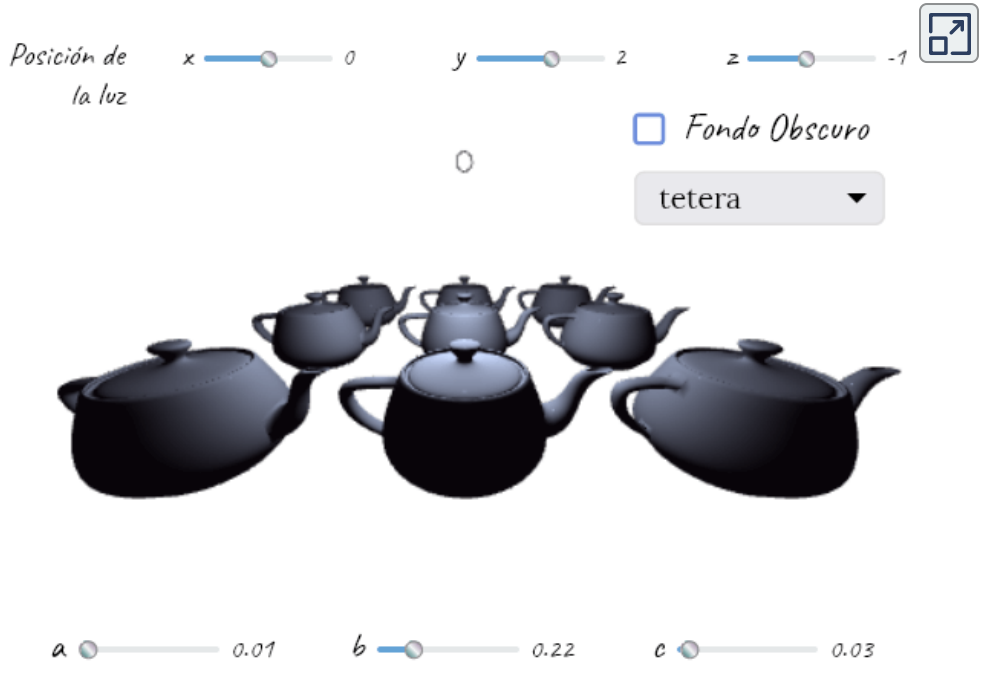


Figura 6.25. Modelos iluminados por una fuente de luz puntual. Modifica la posición de la fuente de luz y los coeficientes del factor de atenuación y observa los resultados. Puedes mover la cámara con el ratón o dedo, así como alejarte o acercarte con la rueda del ratón.

6.7.2 Luz direccional

Una fuente de luz direccional también llamada como *fuente de luz infinita* es un caso especial de luz puntual. Este tipo de luz es utilizada para modelar fuentes de luz que se encuentran a una distancia muy lejana como el sol. Como se muestra en la **Figura 6.26**, visto desde una posición cualquiera en la Tierra los rayos emitidos por el sol son paralelos y su intensidad es prácticamente la misma, es decir, no disminuye con la distancia como en la luz puntual.

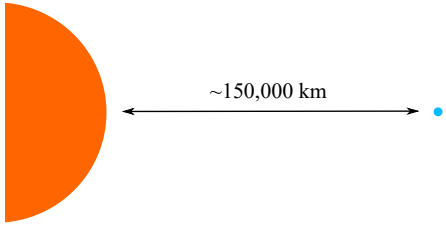


Figura 6.26. El sol como fuente de luz direccional. El ángulo de diferencia entre los polos de la Tierra es de 0.005 grados.

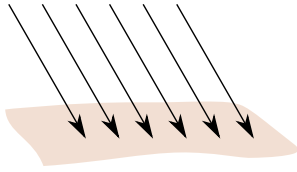


Figura 6.27. Luz direccional. Rayos de luz paralelos.

Como resultado los rayos de luz llegan en una misma dirección de manera uniforme, es decir, que las superficies con la misma orientación reciben la misma cantidad de luz independientemente de su posición. Por lo que la luz direccional puede ser modelada únicamente por sus propiedades de reflexión y su vector de dirección unitario

```
Luz_direccional {
  Vector3 dirección;

  Vector3 ambiental;  //L_A
  Vector3 difuso;     //L_D
  Vector3 especular;  //L_S
}
```

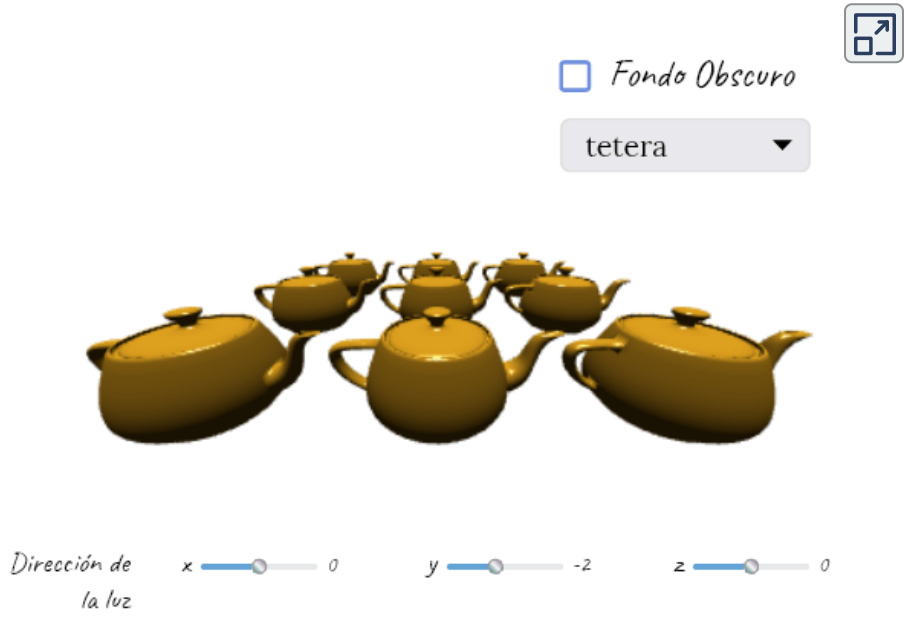


Figura 6.28. Modelos iluminados por una fuente de luz direccional. Modifica la dirección de la luz. Puedes mover la cámara con el ratón o dedo, así como alejarte o acercarte con la rueda del ratón.

6.7.3 Luz de reflector

Una fuente de luz de reflector o *spotlight* es similar a una fuente de luz puntual solo que sus rayos apuntan hacia una dirección específica. Un buen ejemplo de este tipo de fuente de luz son los reflectores de un teatro o un foco envuelto por una lámpara.

Para definir un spotlight necesitamos especificar una posición P en el espacio, que será de donde emitirá energía; una dirección $\mathbf{D}$, que será a la que apunte y a su vez el eje del cono, y un ángulo de corte θ, que es el ángulo formado entre el eje del cono y un rayo a lo largo del borde del cono, veáse **Figura 6.29**. El ángulo de corte define el área a iluminar, fuera de ésta la intensidad de luz es igual a cero. De modo que aquellos puntos que no se encuentren dentro del cono no serán iluminados, limitando así el rango de rayos emitidos por la fuente de luz.

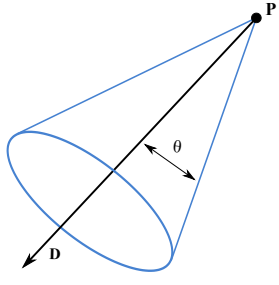


Figura 6.29. Cono formado por una luz de reflector.

Considerando entonces al punto Q sobre un objeto de la escena, podemos verificar si éste debe ser iluminado o no por el reflector de la siguiente forma. Sea γ el ángulo formado entre la dirección de la luz $\mathbf{D}$ y el vector $\mathbf{L}$, que en este caso para facilitar la visualización del cálculo será definido como el vector que va de la fuente de luz a la superficie. Entonces podemos determinar que Q está dentro del cono si γ es menor a θ, por lo que debe ser iluminado, en otro caso se encuentra fuera del cono y no debe ser iluminado, como se muestra en la **Figura 6.30**.

Otra característica importante de este tipo de luz es la atenuación. La intensidad de la luz es atenuada con respecto a la distancia al igual que con una fuente de luz puntual, pero también es atenuada por otro factor, conocido como efecto spotlight.

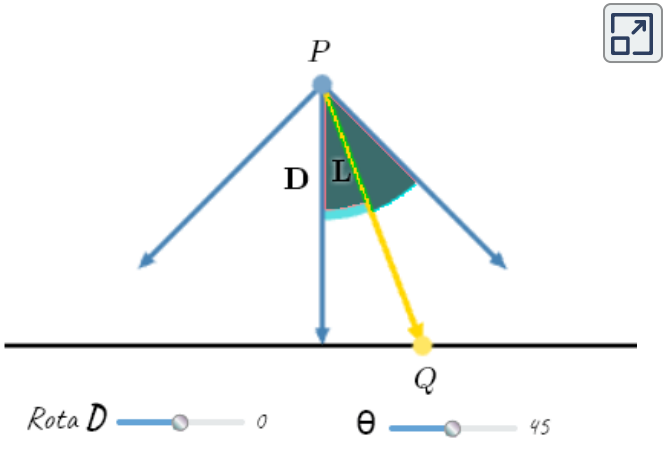


Figura 6.30. Determinación de un punto Q dentro de un reflector. Para iluminar a Q el ángulo de corte θ representado por el sector circular rojo deberá ser mayor o igual, o bien, contener al ángulo γ representado por el arco azul formado entre el vector $\mathbf{L}$ que va de la posición de la fuente al fragmento y el de dirección o eje del cono $\mathbf{D}$. Puedes rotar el cono y modificar el ángulo de corte del cono.

Este efecto consiste en que la intensidad de la luz del reflector decae a medida que los rayos se alejan del eje del cono. Y existen varias formas de modelar este comportamiento. Tomando en cuenta que

$$\cos\gamma = \mathbf{D}\cdot\mathbf{L}$$

Entonces podemos calcular la intensidad de luz recibida por un punto Q como

$$\text{atenuación} = \begin{cases} \dfrac{\max((\mathbf{D}\cdot\mathbf{L}),0)^f}{ad^2+bd+c} & \text{si } \gamma < \theta \\ 0 & \text{en otro caso} \end{cases}$$

donde los coeficientes a, b y c son constantes de atenuación características de la fuente, d la distancia que hay entre la posición de la luz y el punto Q, y $\mathbf{L}$ el vector unitario que va de P a Q.

El exponente f determina el factor por el cual la intensidad es reducida para rayos lejanos al eje del cono. De modo que un valor grande de f nos dará una concentración de luz mayor cerca del eje, mientras que un valor pequeño nos da un haz de luz menos concentrado. La luz es más intensa cuando $\mathbf{D} = \mathbf{L}$ y va decayendo gradualmente a medida que el ángulo θ se incremente.

Entonces la luz de reflector puede ser modelada como sigue

```
Luz_reflector {
  Vector3 posición;
  Vector3 dirección;

  Vector3 ambiental;  //L_A
  Vector3 difuso;     //L_D
  Vector3 especular;  //L_S

  //Coeficientes de atenuación
  float constante;    //a
  float lineal;       //b
  float cuadrático;   //c

  float ángulo_de_corte;   //cos(theta)
  float factor_spotlight; //f factor de atenuación
}
```

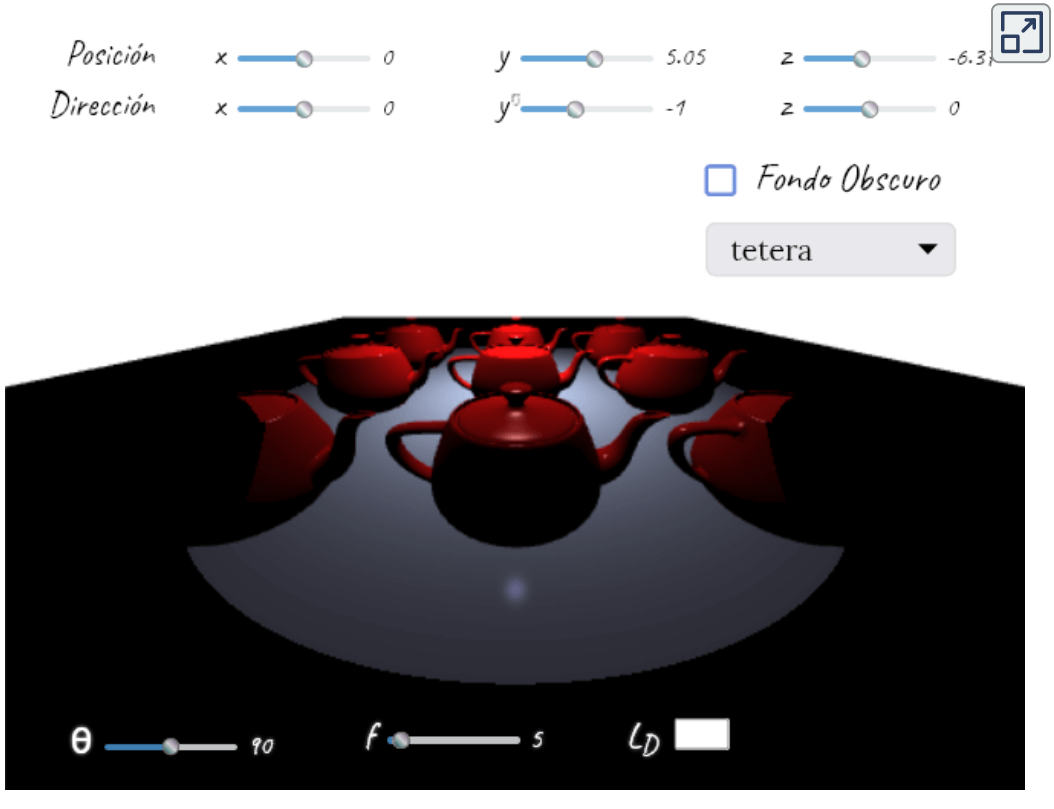


Figura 6.31. Modelos iluminados por una fuente de luz de reflector. Modifica el factor f y el valor de θ, y observa la diferentes concentraciones de luz. Puedes mover la cámara con el ratón o dedo.

Otra manera de modelar el efecto spotlight es seccionando al cono en dos partes: la umbra, que es el área del centro del cono, en la que todos los rayos de luz llegarían y por lo tanto se iluminaría por igual, y la penumbra, que es el área alrededor, en la cual llega una menor cantidad de rayos de luz por lo que se iluminaría menos, ya que suelen ser bloqueados por la superficie que rodea al foco.

Considerando esto podemos definir entonces a la umbra como el área definida por un ángulo ρ, con $\rho < \theta$, y a la penumbra como el área definida entre ρ y θ. Entonces podemos definir la intensidad como

$$I = \begin{cases} 1.0 & \text{si } \gamma < \rho \\ 0.0 & \text{si } \gamma > \theta \\ \left(\dfrac{\cos\gamma - \cos\rho}{\cos\theta - \cos\rho}\right)^f & \text{en otro caso} \end{cases}$$

Finalmente se multiplicaría la atenuación utilizada en la luz puntual por I. Sin embargo, como podemos notar al calcular la intensidad para puntos que están dentro de la penumbra resulta ser algo costosa, en su lugar puede ser reemplazada por otra función, por ejemplo, una interpolación lineal.

6.8 Transparencia

La transparencia es otra característica que está presente en muchos objetos del mundo real. Los materiales transparentes o translúcidos son aquellos que permiten pasar cierta cantidad de luz a través de ellos y hay diferentes maneras en que esto ocurra. Algunos efectos que se pueden producir es que la luz se atenue (transmisión) o se desvíe (refracción), provocando que otros objetos en la escena sean iluminados de manera diferente. Nosotros trataremos el caso más sencillo, en donde el objeto transparente solo atenúa los colores de los objetos que se encuentran detrás de él, preocupándonos únicamente por renderizar al objeto mismo.

La solución más utilizada consiste en mezclar el color del objeto transparente con los colores de los objetos que están atrás de él. Para ello, se agrega una cuarta componente a las tripletas RGB que utilizamos para describir las propiedades reflexivas de nuestros materiales, esta componente es conocida como componente *alfa* (α), y es con la cual describiremos el grado de transparencia en ese punto del objeto. En donde si alfa es igual a 1 quiere decir

que el objeto es opaco, y 0 quiere decir que es totalmente transparente. De este modo, cada fragmento del objeto estará descrito por un color $RGBA$ o $RGB\alpha$.

Recordemos del Pipeline gráfico que el búfer de color terminará almacenando el color final que tendrá el pixel en la pantalla, por lo que el color final deberá ser la mezcla de colores que estamos buscando. Para poder calcular esa mezcla correctamente necesitamos definir un orden de renderizado específico entre los objetos, donde los objetos opacos sean renderizados primero y los transparentes después. Esto es ya que iremos calculando el color final de la siguiente manera.

Para cada fragmento se consideran dos colores: el color del fondo y el color del objeto transparente, en donde el color de fondo es el último color almacenado en el búfer de color. Entonces, el color final $\mathbf{c}_f$ de superponer un objeto transparente en la escena es

$$\mathbf{c}_f = \mathbf{c}_t\alpha_t + (1 - \alpha_t)\mathbf{c}_b$$

en donde $\mathbf{c}_t$ es el color del objeto transparente, α_t el grado de transparencia del objeto, y $\mathbf{c}_b$ el color de fondo actual del búfer. Por lo que el color $\mathbf{c}_b$ es reemplazado por el $\mathbf{c}_f$ en el búfer de color. De esta manera la componente α determinará qué tanto está siendo cubierto el pixel por el material transparente. Por otra parte, si solo se reciben colores $RBG\alpha$ con α igual a 1 la ecuación se simplifica a reemplazar completamente los colores en el búfer de acuerdo a su profundidad.

Este algoritmo nos proporciona una sensación de transparencia en el sentido de que podemos percibir a través de un objeto aquellos objetos que se encuentran detrás de él, de manera similar a como lo haría una tela de gasa. Sin embargo, se sigue sintiendo poco convincente para simular algunos materiales como plástico o vidrio de color.

Supongamos que tenemos un filtro de plástico rojo que cubre a un objeto azul, entonces usando la operación previa, tendríamos como resultado la porción del color rojo y del color azul sumadas. No obstante, como vimos al principio de este capítulo sería mejor multiplicar ambos colores, añadiendo así la luz que refleja el objeto azul hacia el objeto transparente, esto es

$$\mathbf{c}_f = \mathbf{c}_t\alpha_t + \mathbf{c}_b$$

con esta nueva operación (*additive blending* o mezcla aditiva), podemos obtener un mejor resultado para efectos brillantes como chispas o luces, ya que en lugar de atenuar los pixeles, los hace más brillantes. Pero, sigue sin funcionar adecuadamente ya que las superficies opacas no son filtradas, además para superficies semitransparentes como el fuego o humo puede dar un efecto de saturación sobre sus colores. Por estas razones es que la primera operación de mezcla es más utilizada. Puedes observar la diferencia entre estos dos operadores en la **Figura 6.32**.

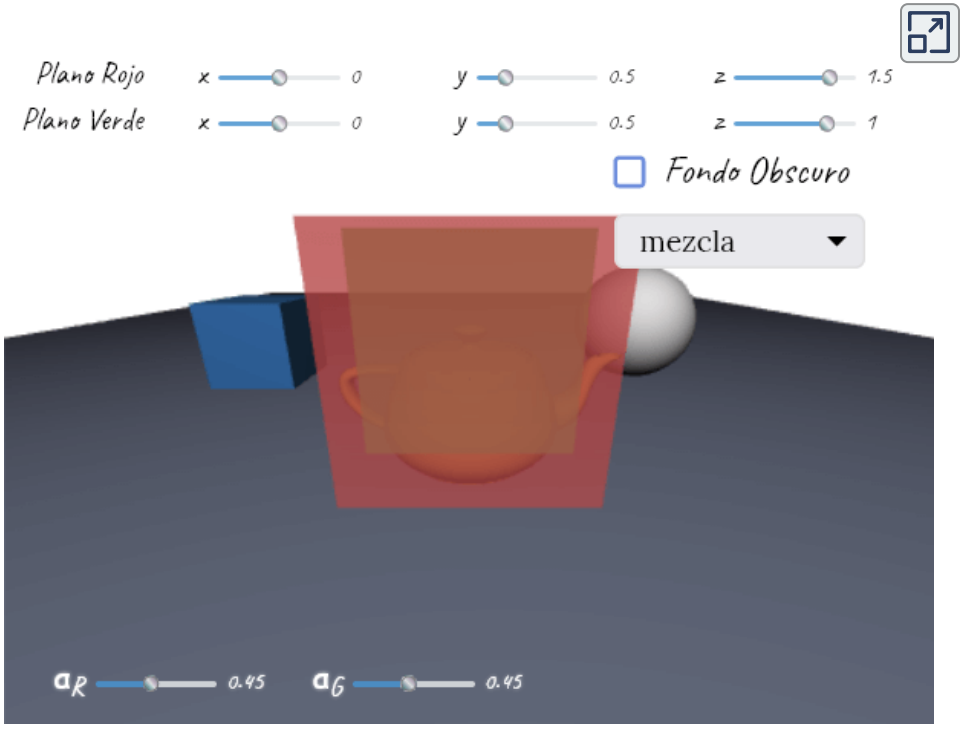


Figura 6.32. Ejemplo de transparencia de dos planos. Se muestran dos planos transparentes. El plano verde es renderizado antes que el plano rojo, intenta encimar o poner el plano verde frente al rojo y observa lo que pasa. Modifica los valores de transparencia α_R y α_G para hacer a los planos de un material más o menos transparente. Cambia el tipo de operador a utilizar y observe las diferencias. Puedes mover la posición de la cámara con el ratón o dedo.

Al renderizar los objetos translúcidos el orden es importante, siendo más específicos necesitamos renderizar las superficies transparentes en un orden de atrás hacia adelante. Una forma de hacer esto es considerando las distancias del centro de los objetos en dirección a la cámara. No obstante esta solución puede no ser trivial en ciertos casos, por ejemplo, cuando los objetos tienen la

misma profundidad. En este caso el operador de mezcla aditiva puede ser una mejor solución, puedes probar esto en la figura anterior.

Otro ejemplo, sería cuando una escena es modificada más adelante, en donde el orden de los objetos cambia, este caso también lo podemos ver en la **Figura 6.32** al mover al plano verde enfrente del rojo. Esto se debe al test del búfer de profundidad, ya que al ser dibujado primero el plano verde y estár más adelante que el rojo, el búfer de profundidad se actualiza y el fragmento del plano rojo es descartado por no ser visible. Por esta razón es que es necesario apagar el test de profundidad durante el renderizado de los objetos transparentes.

Del mismo modo, este problema ocurre si tuviésemos objetos transparentes con concavidades que se sobreponen a sí mismos. Aquí deberíamos prestar especial atención al orden de sus caras. Para mejorar la apariencia en este caso se suele utilizar la orientación de sus caras, y dependiendo de la posición del observador se dibujarían primero las caras traseras del objeto y después las caras frontales, obsérvese la **Figura 6.33**.

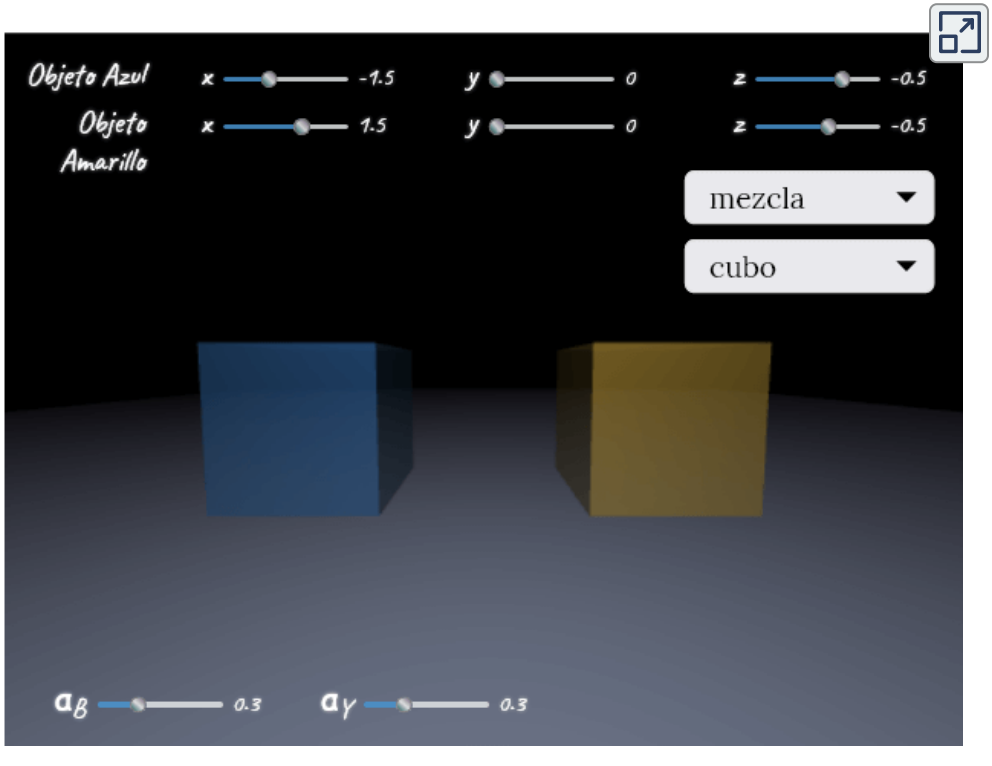


Figura 6.33. Ejemplo de objetos transparentes. Modifica el valor de transparencia de cada objeto y cambia su posición. Puedes mover la posición de la cámara con el ratón o dedo.

6.9 Referencias bibliográficas

[102] Lengye, Eric. ***Mathematics for 3D Game Programming and Computer Graphics.*** Course Technology, a part of Cengage Learning. 3ª Edición. 2012.

[103] Ganovelli, Fabio, et al. ***Introduction to Computer Graphics a Practical Learning Approach.*** CRC Press. 2015.

[104] Shalini Govil-Pai. ***Principles Of Computer Graphics Theory and Practice Using OpenGL and Maya.*** Springer. 1ª Edición. 2005.

[105] Angel, Edward, et al. ***Interactive Computer Graphics a Top-Down Approach with WebGL.*** Pearson Education, Inc. 7ª Edición. 2014.

[106] Dempski, Kelly, et al. ***Advanced Lighting and Materials with Shaders.*** Wordware Publishing, Inc. 1ª Edición. 2005.

[107] Phong, Bui Tuong. Junio de 1975. ***Illumination for Computer Generated Pictures***. *Communications of the* ACM. 18(6): 311-317. https://dl.acm.org/doi/10.1145/360825.360839

[108] ***Introduction to Shading: Lights***. s.f. Scratchapixel. Recuperado el día 12 de Mayo del 2021 de https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-lights

[109] ***Introduction to Shading: Diffuse or Lambertian Shading***. s.f. Scratchapixel. Recuperado el día 12 de Mayo del 2021 de https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/diffuse-lambertian-shading

[110] ***WebGL2 3D - Spot Lighting***. s.f. WebGL2Fundamentals. https://webgl2fundamentals.org/webgl/lessons/webgl-3d-lighting-spot.html

[111] de Vries, Joey. 2014. ***Colors***. LearnOpenGL. https://learnopengl.com/Lighting/Colors

[112] de Vries, Joey. 2014. ***Basic Lighting***. LearnOpenGL. https://learnopengl.com/Lighting/Basic-Lighting

[113] de Vries, Joey. 2014. ***Materials***. LearnOpenGL. https://learnopengl.com/Lighting/Materials

[114] de Vries, Joey. 2014. ***Light casters***. LearnOpenGL. https://learnopengl.com/Lighting/Light-casters

[115] de Vries, Joey. 2014. ***Multiple lights***. LearnOpenGL. https://learnopengl.com/Lighting/Multiple-lights

[116] Luebke, David. 2011. ***Lighting & Shading***.[Diapositiva de PowerPoint] COMPSCI 464: Computer Graphics. http://cs.boisestate.edu/~alark/cs464/lectures/Shading.pdf

[117] ***Phong reflection model***. 27 de Diciembre del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Phong_reflection_model&oldid=996573952

[118] ***The Phong Model, Introduction to the Concepts of Shader, Reflection Models and BRDF***. Scratchapixel. Recuperado el día 12 de Mayo del 2021 de https://www.scratchapixel.com/lessons/3d-basic-rendering/phong-shader-BRDF

[119] ***Reflexión y Refracción de la Luz***. Fisicalab. Recuperado el día 12 de Mayo del 2021 de https://www.fisicalab.com/apartado/reflexion-refraccion-luz

[120] Brown, C. Wayne. 4 de Marzo del 2016. ***9.3 - Specular Lighting***. Learn WebGL. Recuperado el día 12 de Mayo del 2021 de http://learnwebgl.brown37.net/09_lights/lights_specular.html

[121] Thormählen, Thorsten. Kehrer, Johannes. s.f. ***Phong demo***. Toronto - CS. Recuperado el día 13 de Mayo del 2021 de http://www.cs.toronto.edu/~jacobson/phong-demo/

CAPÍTULO VII

Texturas



Figura 7.1. Ejemplo de mapeo de textura sobre una esfera. Texturas del planeta Tierra obtenidas de https://www.solarsystemscope.com/ bajo la Licencia CC BY 4.0.

En el capítulo anterior asumíamos que las propiedades reflectivas de un objeto eran las mismas para cada punto de su superficie, sin embargo, muy pocas superficies son así.

Por ejemplo, si quisiéramos representar una superficie plana con un patrón de colores sencillo como el de un tablero de ajedrez, podríamos utilizar $8 \times 8 = 64$ cuadrados ordenados, alternando los materiales (blanco y negro) entre cada uno. Pero si consideramos un patrón de colores más complicado como un texto, una pintura o propiedades de materiales naturales como la madera y el mármol, se vuelve muy difícil de representar geométricamente, además de ser excesivamente caro de almacenar ya que se requeriría de un gran número de polígonos para tener un acabado más realista.

Del mismo modo, una superficie también puede presentar otro tipo de imperfecciones como rasgaduras, hendiduras, bultos y variaciones de colores, las cuales presentarían este mismo problema.

Las técnicas de mapeo de texturas que describirémos en este capítulo nos permitirán controlar las propiedades reflectivas de cada punto de una superficie, sin necesidad de agregar geometría extra en la mayoría de los casos, logrando hacer que nuestras superficies se vean más realistas en tiempo real.

7.1 Pipiline de texturas

El mapeo de texturas o *texture mapping* es una técnica que nos permite modelar de manera eficaz las variaciones de material y acabado de una superficie. Dicha técnica consiste en modificar las propiedades reflectivas y otros atributos de cada punto de la superficie utilizando los valores almacenados en uno o varios *mapas de textura*, los cuales suelen ser una imagen o una función procedural.

Los mapas de textura o simplemente texturas, son espacios de 1, 2, 3 o 4 dimensiones, en donde se almacena la información que queremos mapear a una superficie. Para acceder a los valores de una textura necesitamos especificar sus *coordenadas de textura*. Por ejemplo, una textura 1D puede usarse para determinar la coloración de un modelo de terreno de acuerdo a la altura de cada punto, o bien, para pintar una curva o una línea de un color a otro dado su parámetro t.

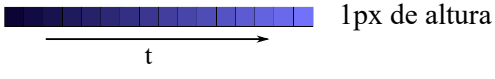


Figura 7.2. Ejemplo de un mapa de textura 1D.

Las texturas 2D o de imágenes por otra parte son las más utilizadas, y se trata de imágenes rasterizadas, las cuales son representas por arreglos bidimensionales de pixeles. Podemos pensarlas como una etiqueta que es "pegada" a lo largo de una superficie.

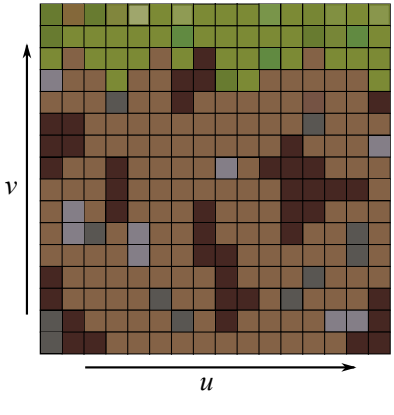


Figura 7.3. Ejemplo de un mapa de textura 2D.

Las texturas 3D o de volumen son descritas por un arreglo tridimensional de píxeles; sus coordenadas de textura son presentadas como vectores (u, v, w), donde w hace referencia a la profundidad del arreglo. Podemos pensarlas como un arreglo de texturas 2D, como se observa en la **Figura 7.4**. Y debido a que suelen ser muy costosas de almacenar usando imágenes, se suele optar por usar texturas procedurales en su lugar, en donde las coordenadas son mapeadas por una función a un color.

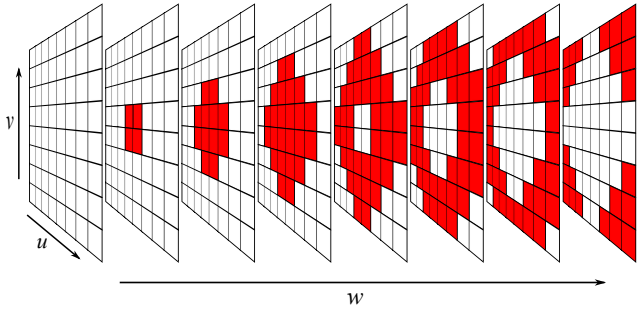


Figura 7.4. Ejemplo de un mapa de textura 3D.

La **Figura 7.5** muestra las diferentes etapas para aplicar una textura a una superficie, las cuales se describen a continuación. *REF*. [122].

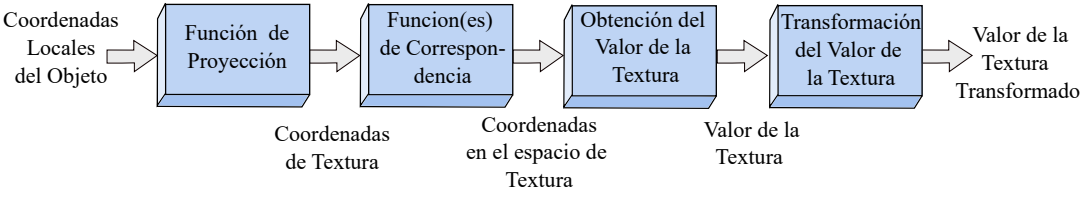


Figura 7.5. Pipeline general de texturizado .

7.1.1 Función de proyección

El proceso inicia en la definición de las posiciones de los vértices de algún modelo, es decir, desde el espacio local, donde por medio de una *función de proyección* las coordenadas de textura son asignadas a los vértices como un atributo.

La función de proyección se encarga de mapear o proyectar un punto tridimensional al espacio de textura, en otras palabras, cada punto de la superficie deberá tener asignado un punto en la textura.

Por lo regular este mapeo es a un espacio 2D, también conocido como espacio UV o ST, el cual se encuentra definido dentro del rango $[0, 0] \times [1, 1]$, nosotros nos referiremos a sus coordenadas simplemente como coordenadas uv. Comúnmente los programas de modelado permiten a los artistas definir las coordenadas de textura uv por vértice, las cuales pueden ser inicializadas a partir de funciones de proyección o algoritmos de desenvolvimiento de malla, además de permitir ser editadas de manera manual. Los algoritmos de desenvolvimiento de malla pertenecen a un campo amplio llamado *mesh parametrization*. Algunas de la funciones de proyección más utilizadas son las cilíndricas, esféricas y planas, ver **Figura 7.6**.



Figura 7.6. Funciones de proyección. Selecciona el tipo de proyección que quieras visualizar, así como la textura. Puedes mover la posición de la cámara con el ratón o dedo y acercarte o alejarte con la rueda del ratón.

La especificación de la función de proyección también depende de cómo esté definido el modelo, por ejemplo, en una superficie parámetrica las coordenadas uv son obtenidas por definición. Las texturas volumétricas o 3D por otra parte suelen utilizar las mismas coordenadas del modelo como coordenadas de textura.

En el renderizado en tiempo real las funciones de proyección normalmente son aplicadas en la etapa de modelado y las coordenadas de textura almacenadas en los vértices.

7.1.2 Función de correspondencia

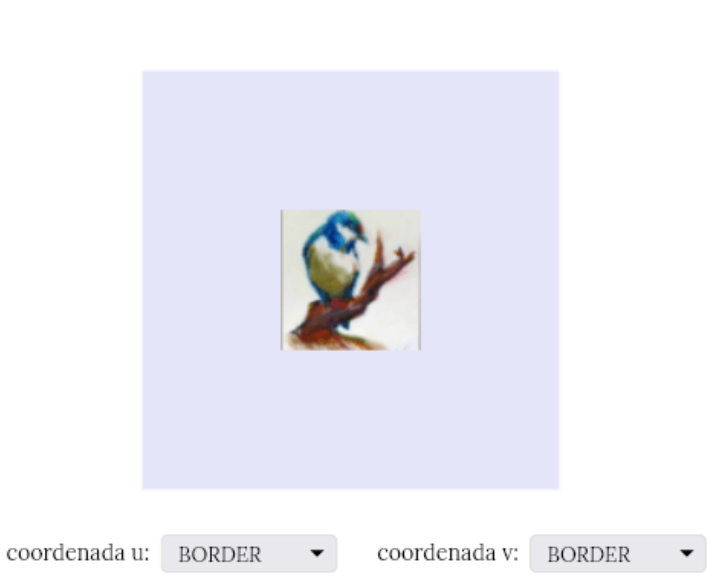


Figura 7.7. Texture wrapping modes. Las coordenadas de textura del plano están definidas dentro del rango $[-1, 2] \times [-1, 2]$, nótese que el punto $(0, 0)$ se encuentra en la esquina inferior izquierda del cuadro central. Selecciona las diferentes funciones de correspondencia para cada coordenada y observa el resultado.

Una función de correspondencia convierte las coordenadas de textura en coordenadas en el espacio de textura, permitiendo una mayor flexibilidad durante la aplicación de la textura.

Por otra parte, ya que el espacio de textura 2D es un espacio rectangular definido dentro del rango $[0, 0] \times [1, 1]$, es importante especificar qué ocurre cuando nos salimos de este rango para evitar lidiar con excepciones. Para ello, es necesario especificar una función de correspondencia que mapee las coordenadas de $[-\infty, -\infty] \times [\infty, \infty]$ a $[0, 1]^2$ de cierto modo, a este tipo de funciones se les conoce como *texture wrapping* y algunas de las más comúnes son descritas a continuación:

- *clamp*: el resultado es la repetición de los bordes de la imagen para las coordenadas fuera del rango.
- *repeat*: la imagen se repite a lo largo de la superficie, esto se hace eliminado la parte entera de la coordenada.
- *mirror*: la imagen se repite a lo largo de la superficie, pero es invertida en cada una de las repeticiones.
- *border* (DirectX): las coordenadas fuera del rango son renderizadas con un color previamente específicado, formando así un borde.

cada una de las funciones puede ser aplicada en ambas coordenadas u y v, o bien definir un modo distinto para cada una. En la **Figura 7.7** puedes probar los diferentes modos de texture wrapping.

Otro ejemplo sería aplicar una transformación geométrica sobre las coordenadas de textura en el *vertex shader* o *fragment shader*, como se muestra en la **Figura 7.8**, o bien, utilizar la API para definir un área de la textura sobre la cual se trabajará.

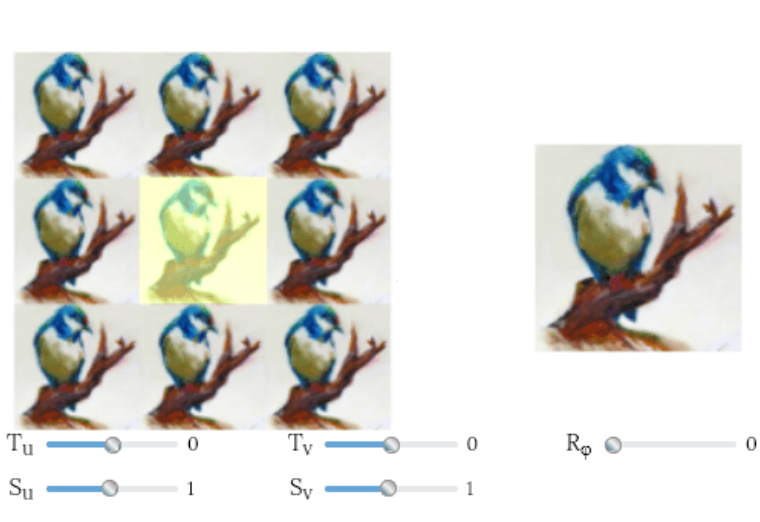


Figura 7.8. Transformaciones sobre las coordenadas de textura. La ilustración de la izquierda muestra el espacio de textura dentro del rango $[-1, 2] \times [1, 2]$ con el modo *REPEAT* de texture wrapping, el rectángulo amarillo está definido por las coordenadas de textura del plano renderizado que se ilustra a la derecha, señalando el área de la textura que se va a mapear sobre el plano de la derecha. Las coordenadas de textura son inicializadas en $[0, 1]^2$. Aplica las transformaciones sobre las coordenadas de textura y obsérva cómo el rectángulo amarillo es transformado, afectando la imagen que es proyectada sobre el plano de la derecha.

La última función de correspondencia es implícita y consta de multiplicar las coordenadas de textura dentro del rango $[0, 1]$ por la resolución de la imagen, para obtener así la ubicación del pixel de la textura.

7.1.3 Valores de la textura

Finalmente con las coordenadas en el espacio de textura podemos obtener o acceder a los valores del mapa de textura. Como mencionamos anteriormente, los mapas de texturas pueden ser una imagen o una función procedural. En el caso de ser una imagen, el valor es obtenido del pixel de la imagen correspondiente a las coordenadas de textura, este proceso es explicado en el siguiente tema. Mientras que para una función procedural, conocida como texturas procedurales, basta realizar el cálculo de la función asociada como textura.

El valor obtenido del mapa suele ser una tripleta RGB, la cual puede ser utilizada para reemplazar el color de la superficie. De manera similar, se puede obtener un color en escala de grises (ya que se utiliza el mismo valor en las tres componentes) para representar la intensidad de la componente especular, o bien, utilizar el color para representar una normal como veremos más adelante, entre otros valores que se quieran utilizar para modificar la ecuación del modelo de iluminación. Otro valor que puede obtenerse es la 4-tupla $RGBA$ donde A representa la transparencia del color, la cual suele ser utilizada como máscara para definir los pixeles a considerar y a descartar de la textura.

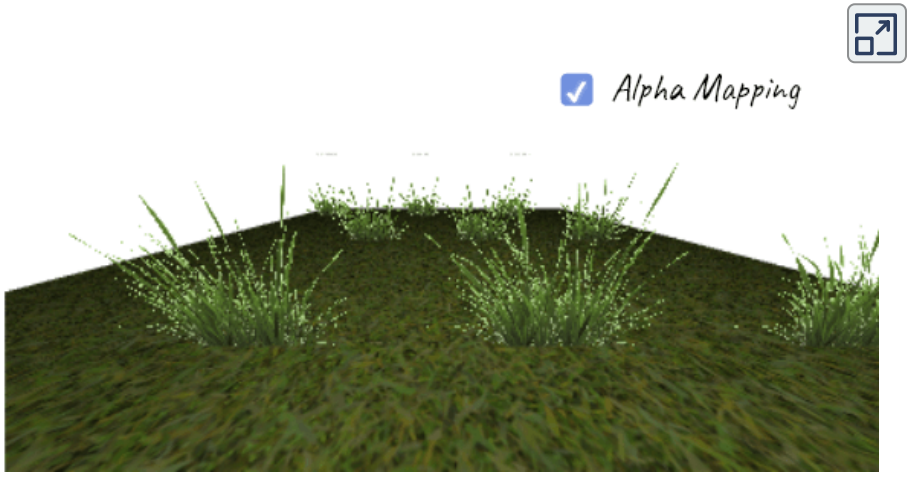


Figura 7.9. *Alpha Mapping*. En este ejemplo los valores la textura (texeles) con $A < 0.5$ son descartados. Puedes mover la posición de la cámara con el ratón o dedo. Texturas de pasto obtenida de https://learnopengl.com/ bajo la Licencia CC BY 4.0.

7.1.4 Texturas en el pipeline gráfico

Por otro lado, volviendo al Pipeline gráfico, el mapeo de texturas es realizado principalmente dentro de la etapa de Rasterización y la primera del procesamiento de fragmentos, *Pixel shading*. Ya que durante la primera las coordenadas de textura (y otros atributos) que han sido asignados a los vertices de las primitivas son interpoladas entre sí para obtener las coordenadas de textura correspondiente a cada fragmento. Y en la segunda se determinaría el color del fragmento usando la ecuación de iluminación que se desee con los valores obtenidos del mapa de texturas dadas sus coordenadas de textura.

7.2 Texturas 2D o de imágenes

Una textura 2D es una imagen rasterizada o mapa de bits, el cual es representado por un arreglo bidimensional de pixeles. Los pixeles de la imagen son llamados como texeles (por *texture element* o *texture pixel*) para diferenciarlos de los pixeles de la pantalla. El mapeo de una imagen a una superficie 3D es una tarea bastante compleja ya que la mayoría de veces no se puede obtener el color del texel de manera directa con las coordenadas en el espacio de textura. En esta sección nos enfocaremos en los principales problemas a los que nos enfrentamos para obtener un valor de una imagen y los métodos que se emplean para solucionarlo.

Recordemos que las coordenadas de textura (u, v) han sido asignadas a los vértices, y que son mapeadas por una función de correspondencia al espacio $[0, 1]^2$. Ahora bien, dependiendo del API el origen del sistema puede estar ubicado en la esquina inferior izquierda de la textura, es decir, su esquina inferior es $(0, 0)$ y su esquina superior es $(1, 1)$, como es el caso de OpenGL, o bien, la esquina superior izquierda de la imagen es $(0, 0)$ y su esquina inferior derecha es $(1, 1)$, como es el caso de DirectX; nosotros usaremos el primer sistema de referencia. De modo que no importa el tamaño de la imagen siempre podremos acceder a cada texel variando los valores de u y v.

Supongamos que queremos mapear una imagen de $n \times n$ a un cuadrado y que el cuadrado proyectado en la pantalla es del mismo tamaño que la textura. Entonces tendríamos una correspondencia uno a uno entre los pixeles que conforman la superficie (fragmentos) y texeles, por lo que podríamos acceder a

los valores del arreglo sin problemas, es decir, al fragmento con coordenadas (u, v) le correspondería el texel `imagen[u*n-1][v*n-1]`.

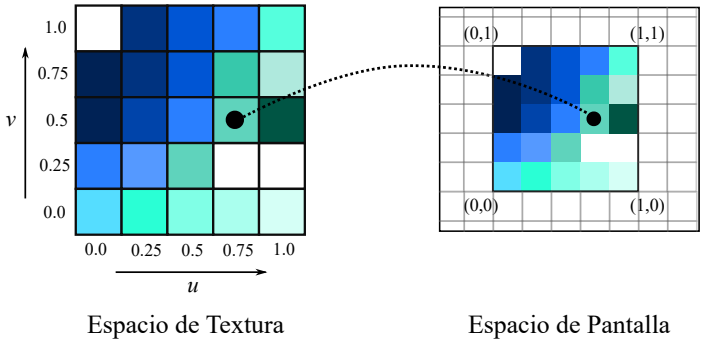


Figura 7.10. Mapeo directo de una textura de 5×5 pixeles sobre un cuadrado que ocupa 5×5 pixeles en la pantalla. Cada vértice del cuadrado está asociado a las coordenadas de textura que se muestran.

Vale la pena mencionar que la posición de los vértices no tienen que tener una relación en particular con las coordenadas de textura. En la **Figura 7.10** se muestra un ejemplo de este caso, en donde se busca mapear un área de la imagen a una primitiva, aquí los vértices de la primitiva tienen asignadas las coordenadas de textura que conforman el triángulo naranja en la imagen.

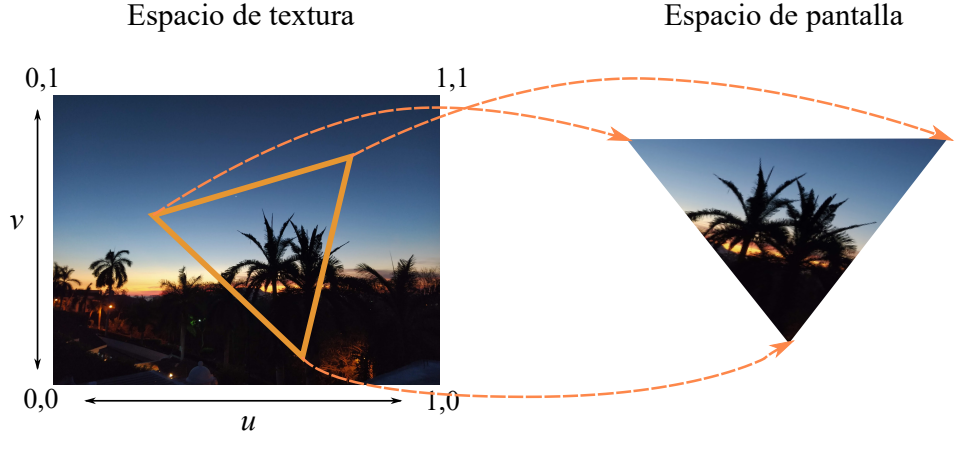


Figura 7.11. Mapeo de una textura a una primitiva.

Nótese que el triángulo de la primitiva tiene una forma diferente que el triángulo formado por las coordenadas de textura, lo cual provoca que la imagen se deforme un poco.

Desafortunadamente, cuando una textura es aplicada sobre una superficie los texeles no suelen tener esta relación de correspondencia uno a uno con los fragmentos de la superficie, como vimos en el primer ejemplo. Y además sabemos que los texeles tienen coordenadas enteras, por lo que no podemos acceder a valores intermedios. Entonces, ¿qué pasa con las coordenadas que no intersecan exactamente con un pixel? La respuesta dependerá del tipo de método de filtro que se seleccione para solucionar cada caso.

La proyección de un pixel al espacio de textura es un cuadrilátero el cual puede ser más pequeño o bien, el pixel puede llegar a cubrir varios texeles. En el primer caso, necesitamos aumentar o estirar la imagen por lo que se deberá aplicar un filtro de *magnificación* (**Figura 7.12** (izquierda)), mientras que en el segundo, necesitamos encoger la textura, por lo que se aplica un filtro de *minimización* (**Figura 7.12** (derecha)).

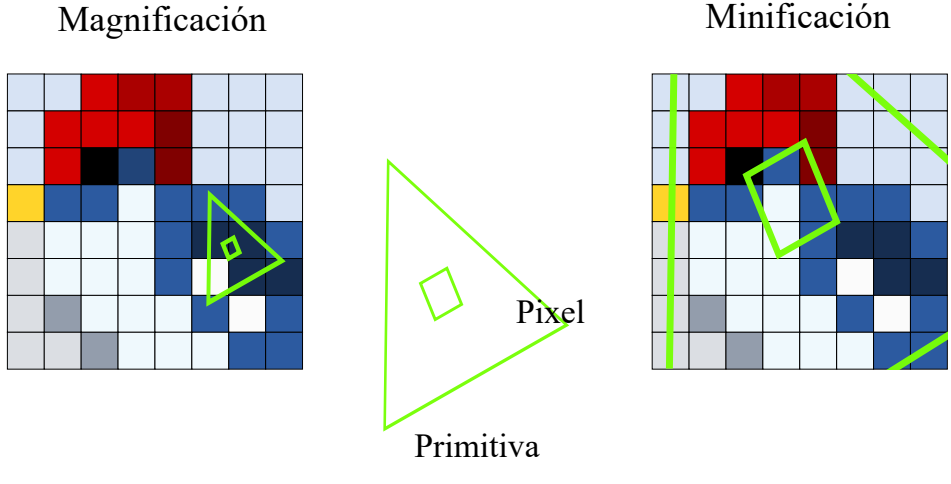


Figura 7.12. Métodos de filtrado: Magnificación y Minimización.

7.2.1 Magnificación

Las dos técnicas de filtrado más comunes para resolver este problema son el filtro caja o *nearest neighbor* y el filtro bilineal o de interpolación bilineal.

En el filtro caja para cada pixel se toma el texel más cercano a su centro. En esta técnica los texeles individuales se hacen más evidentes y es el típico efecto de pixelado que obtenemos al ampliar una imagen, como se muestra en la **Figura 7.13**. Aunque termina dando un resultado de baja calidad, el cálculo es muy rápido pues solo se requiere buscar un texel por pixel.

Por otro lado, en la interpolación bilineal, por cada pixel se toman los cuatro texeles más cercanos a su centro con los cuales se realiza una interpolación bilineal, obteniendo así un valor combinado para el pixel. Con esta técnica el resultado es una imagen más borrosa, disminuyendo considerablemente el efecto de pixeleado que produce la técnica anterior, véase la **Figura 7.13**.

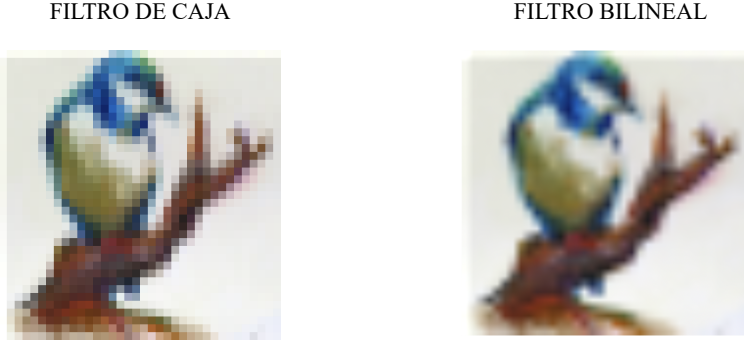


Figura 7.13. Comparación entre filtro de caja y bilineal utilizando una imagen de 36×36 pixeles.

Por ejemplo, supongamos que tenemos las coordenadas en el espacio de textura de un fragmento $(P_u, P_v) = (61.24, 52.18)$. Para encontrar a sus cuatro vecinos más cercanos necesitamos primero restarle $(0.5, 0.5)$ a las coordenadas para estar en el centro del pixel, teniendo (60.74, 51.68). Ahora, considerando solo la parte entera del centro del pixel, es decir con $(x, y) = (60, 51)$, podemos encontrar a los cuatro texeles cercanos $(x, y), (x + 1, y), (x, y + 1)$ y $(x + 1, y + 1)$ como se muestra en la **Figura 7.14**.

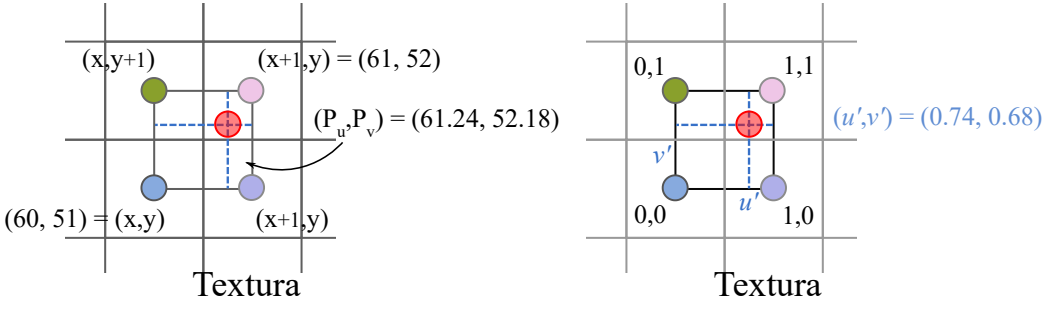


Figura 7.14. Magnificación. Interpolación bilineal.

Finalmente podemos calcular la interpolación bilineal entre los cuatro texeles y como parámetro dentro de esta relación la parte fraccionaria del centro del pixel es decir $(u', v') = (0.74, .61)$.Esto es

$$\begin{aligned}\mathbf{b}(P_u, P_v) =&(1-u')(1-v')C(x,y) + u'(1-v')C(x+1,y)+\\ &(1-u')v'C(x,y+1) + u'v'C(x+1,y+1)\end{aligned}$$

donde $C(x, y)$ es la función que te devuelve el color del textel de la imagen, es equivalente a `imagen[x][y]`. Como podemos notar el color del texel más cercano a las coordenadas (u', v') o centro de las coordenadas en el espacio de textura son las que tendrán mayor influencia en el valor final.

Si bien esta técnica suele dar un mejor resultado que la anterior, puede no ser muy buena opción para ciertos patrones como el del tablero de ajedrez, ya que se mezclarían los colores dando una escala de grises entre cada transición de color, véase **Figura 7.15**.

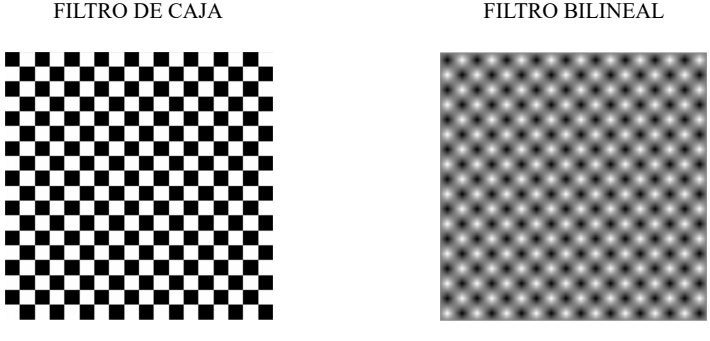


Figura 7.15. Magnificación de una textura con patrón de tablero de ajedrez: Filtro de caja y filtro bilineal.

7.2.2 Minimización y Mipmapping

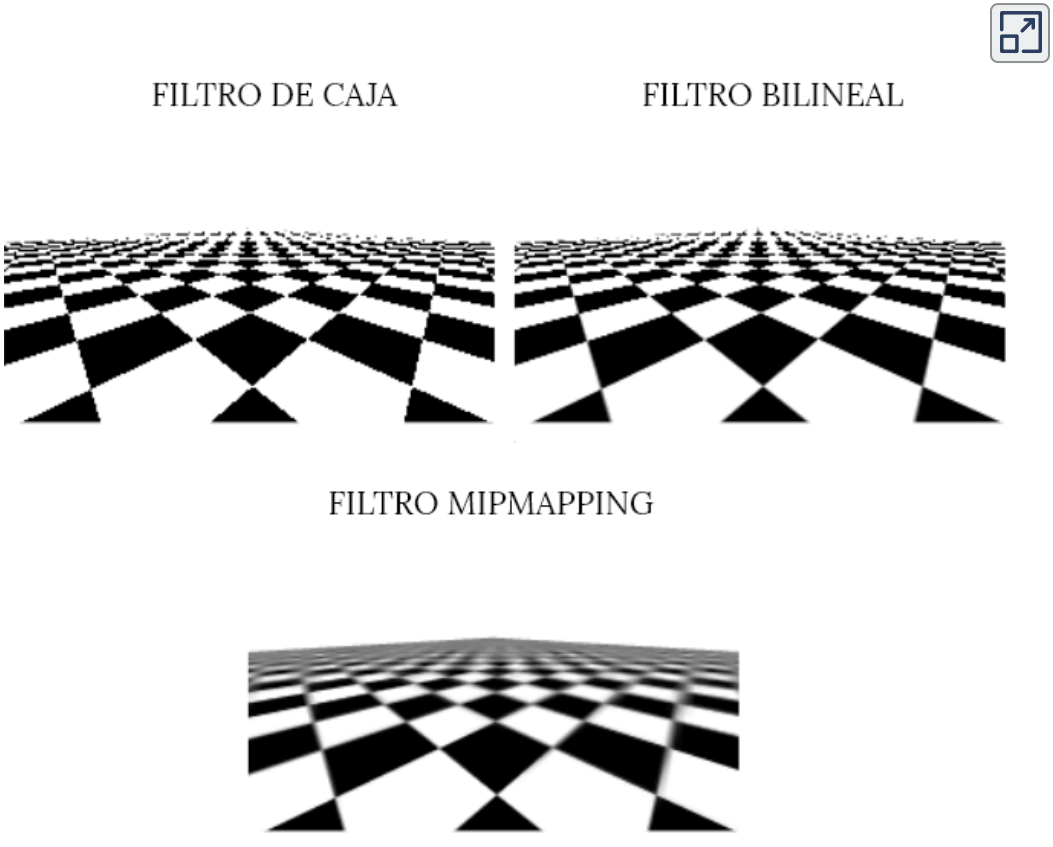


Figura 7.16. Minimización con filtro de caja, filtro bilineal y mipmapping.

Cuando una textura es minimizada, varios texeles de la textura corresponden a un pixel o fragmento. De modo que para obtener el color correcto de cada fragmento deberíamos calcular la influencia que tienen cada uno de los texeles que éste cubre, lo cual resulta imposible de calcular de manera precisa en tiempo real.

Si utilizamos el filtro de caja se tomarían texeles arbitrariamente lejanos entre sí, pudiendo producir un mapeo sin sentido. Como se puede ver en la **Figura 7.16** (izquierda superior) este filtro puede causar muchos problemas de *aliasing* o efecto escalera ya que solo un texel es elegido entre varios, estas distorsiones son más notables cuando la superficie se mueve con respecto al observador y a esto se le conoce como *temporal aliasing*.

Por otra parte, el filtro bilineal es ligeramente mejor ya que en lugar de tomar un texel combina cuatro, sin embargo cuando el pixel llega a cubrir más de cuatro texeles el filtro comienza a fallar de manera similar al anterior y produce aliasing, como se muestra en la **Figura 7.16** (derecha superior).

El método más utilizado que ayuda a mitigar este problema de manera eficiente es el de *mipmapping*, donde *mip* proviene de la frase en latín "*multum in parvo*" que quiere decir "muchas cosas en un lugar pequeño". En este método de filtrado se genera un conjunto de versiones más pequeñas de la imagen original, cada una un cuarto más pequeña que la anterior antes de que comience el renderizado.

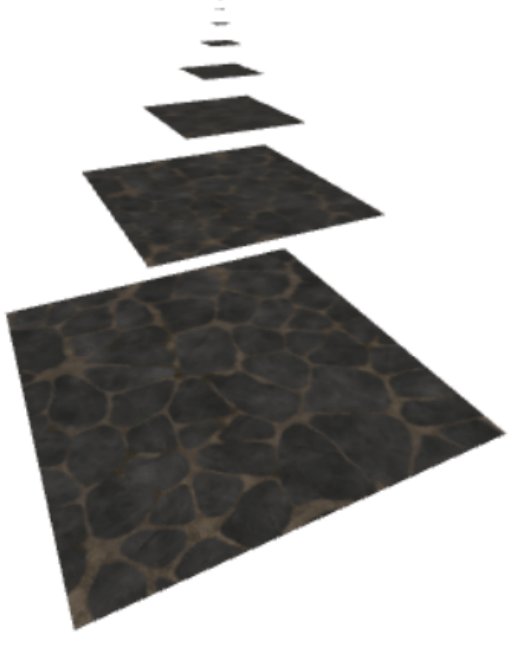


Figura 7.17. Pirámide de mipmaps. Textura obtenida de ambientCG.com bajo la Licencia CC0 1.0 Universal.

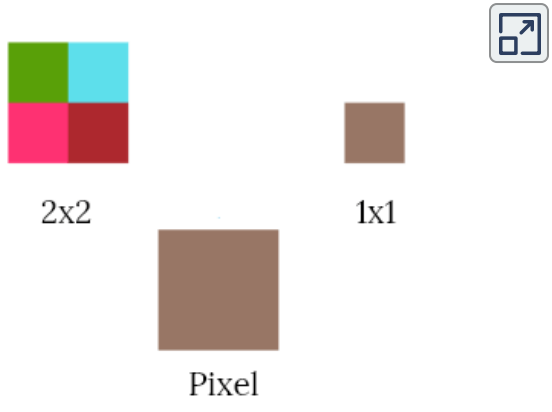


Figura 7.18. Creación de mipmap con una textura con 2×2 texeles. El pixel cubre exactamente los cuatro texeles de la textura, por lo que el nuevo mipmap contará de una textura con un solo texel, y como valor tendrá el promedio de los cuatro texeles de la textura original.

Consideremos el caso ideal, donde tenemos una textura de 2×2 texeles y la proyección de un pixel cubre exactamente estos cuatro texeles. Podemos crear entonces una versión más pequeña de la imagen hecha sólo con un texel, el cual tendrá como color el valor promedio de los cuatro texeles de la textura original, por lo que este valor será el que nos interesará acceder como se ilustra en la **Figura 7.18**.

Este proceso se puede generalizar para una imagen con un ancho w y altura h donde el fragmento cubra un área arbitraria de texeles. De modo que cuando se cree la textura, también se deberán crear las versiones más pequeñas de la misma, reduciendo $\frac{1}{2}$ el ancho y alto de manera recursiva hasta tener una textura con al menos una de sus dimensiones igual a 1, como se muestra en la **Figura 7.17**. Cada una de las imágenes creadas se les llama *mipmap* y es asociada a un nivel, donde el primer nivel es el 0 y está asociado a la textura original, el nivel 1 para el primer mipmap y así sucesivamente. Al conjunto de mipmaps se le conoce como *cadena de mipmaps* o *pirámide de mipmaps*.

Nótese que para construir la piramide de mipmaps como hemos visto, se requiere que la textura original tenga dimensiones de potencia de dos (POT), es decir, $w = 2^n$ y $h = 2^m$. También como pudimos observar los mipmaps se vuelven pequeños muy rápido, ya que se van encogiendo un cuarto de su área cada vez, el total de memoria utilizada es solo un tercio más que la memoria utilizada por la textura original.

Por otro lado, la mayoría de las GPUs modernas ya permiten manejar texturas con dimensiones diferentes a potencias de dos (*NPOT*). Para ello, se suele escalar las dimensiones de la textura a la siguiente potencia de dos más cercana antes de empezar a generar los mipmaps, por lo que se necesita una mayor cantidad memoria y hace que su manejo sea más lento. Por estas razónes es que se recomienda en general usar imágenes con dimensiones iguales a potencia de dos. Además de que algunas GPUs (inlcluyendo modernas) siguen teniendo problemas para mostrar texturas con dimensiones NPOT, como generar alguna distorsión en la imagen original al ser escalada.

Ahora bien, cuando se está renderizado, y el proceso de texturizado se esté ejecutando, para cada fragmento se elige el nivel del mipmap a utilizar calculando qué tan grande es la proyección del pixel, es decir, qué tantos texles cubre de la textura original. Por ejemplo, si el pixel cubre cuatro texeles, entonces se utiliza el mipmap del nivel 1, si cubre 16 se elige el del nivel 2 y así sucesivamente. Para tener mejores resultados se realiza un filtrado trilineal entre los dos mipmaps más cercanos y se interpolan sus dos valores obtenidos.

Para calcular el nivel del mipmap λ a utilizar en un fragmento, también referido como *nivel de detalle*, necesitamos calcular primero el número de texeles que cubre el pixel, siendo $\rho = texeles/pixel$. Una forma es calculando el borde de mayor tamaño del cuadrilátero formado por la proyección del pixel en el espacio de textura. Sea (x, y) las coordenadas del pixel (coordenadas de pantalla) y (u, v) las coordenadas en el espacio de textura del pixel, siendo más específicos, expresado en texeles $([0, 0] \times [2^n, 2^m])$, entonces podemos calcular ρ como

$$\rho = \max\left(\left\|\begin{bmatrix}\frac{\partial u}{\partial x}\\ \frac{\partial v}{\partial x}\end{bmatrix}\right\|, \left\|\begin{bmatrix}\frac{\partial u}{\partial y}\\ \frac{\partial v}{\partial y}\end{bmatrix}\right\|\right)$$

donde cada diferencial es la razón del cambio en la textura con respecto a un eje de la pantalla. Por ejemplo $\frac{\partial u}{\partial x}$ es qué tanto cambió la coordenada de textura u a lo largo del eje x para un pixel, obsérvese la **Figura 7.19**.

Finalmente podemos obtener el nivel de detalle como

$$\lambda = log_2 \rho$$

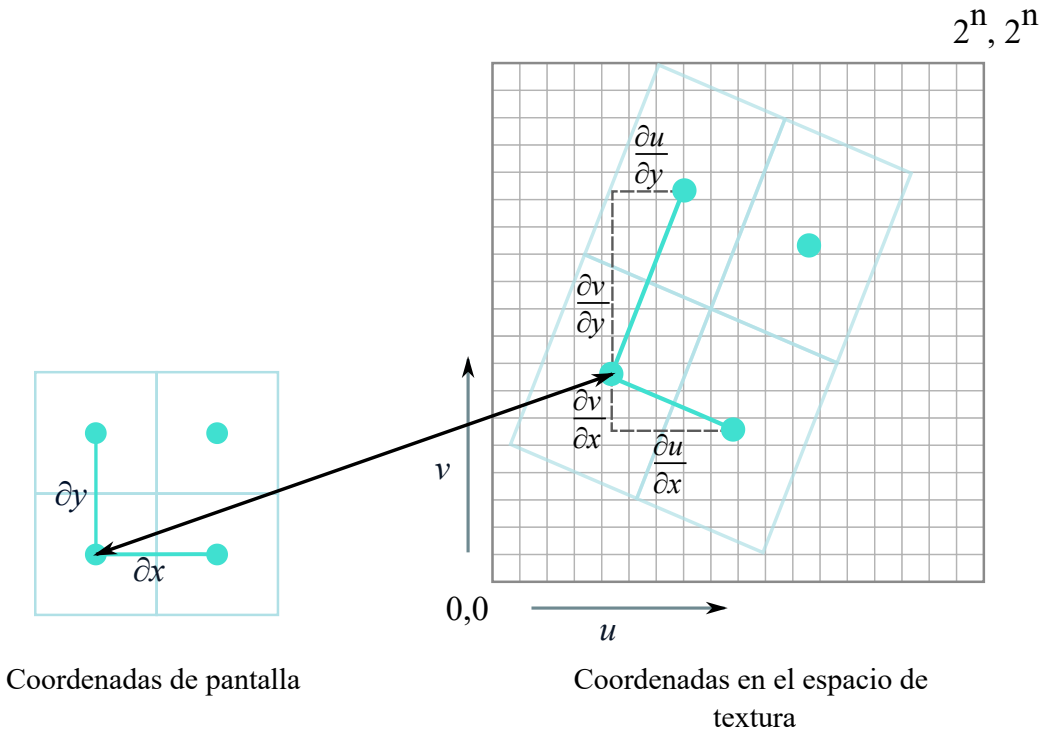


Figura 7.19. Estimación del tamaño de un pixel en el espacio de textura.

O bien, para un mejor resultado podemos hacer una interpolación trilineal entre los dos nivles más cercanos $\lfloor\lambda\rfloor$ y $\lfloor\lambda\rfloor + 1$. En donde primero se hacen dos interpolaciones bilineales en cada uno de los mipmaps usando las coordenadas de textura y por último los valores obtenidos de los mipmaps son interpolados linealmente con la parte flotante de λ.

De este modo, no importa la cantidad de texeles que un pixel cubra, siempre se tardará la misma cantidad de tiempo en calcular su color. Pues en lugar de promediar todos los texeles contenidos por pixel, se interpolan conjuntos de texeles ya precalculados. No obstante, este método sigue presentando algunos defectos. Uno de los más importantes es el exceso de desenfoque, y esto ocurre debido a que solo podemos acceder a áreas cuadrádas de la textura.

Digamos que un framento cubre un gran número de texeles en dirección de u y muy pocos en dirección de v, por lo que nos interesa un área rectangular en la textura. Esto sucede cuando miramos a lo largo de una superficie formando un ángulo oblicuo. E incluso podría llegar a ser necesario hacer una minificación a

lo largo de un eje, y una magnificación a lo largo del otro. Podemos observar este defecto en las líneas que se dezplazan en la distancia de la **Figura 7.16**. El filtro más común para corregir este defecto es el *Anisotropic Filtering*, el cual reutiliza la piramide de mipmaps.

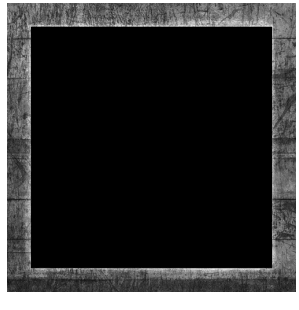
7.3 Mapeo de materiales

Empecemos con uno de los usos más comúnes y sencillos del mapeo de texturas. Como mencionamos anteriormente, podemos usar una textura para modificar las propiedades de los materiales que conformen la superficie, específicando qué áreas de la superficie tienen qué material. De este modo cuando se vaya a evaluar la ecuación de la iluminación del punto podrémos modificar sus parámetros con la información especificada en el mapa de textura.

El párametro que más se suele modificar es el del color de la superfie, donde se reemplaza el color difuso por el valor del mapa, a esta textura se le conoce como *diffuse map* o *abeldo map*. Otro mapa que también suele ser utilizado junto con el diffuse map, es el mapa especular o *gloss map*, que es un mapa en escala de grises usado para modificar la insensidad del componente especular, dando así un efecto más realista. Por ejemplo, en la **Figura 7.21**, se utlizan ambos mapas (**Figura 7.20**) para describir una caja de madera con partes metálicas.



Mapa de textura difuso



Mapa de textura especular

Figura 7.20. Mapa difuso y mapa especular de una caja. Texturas obtenidas de https://learnopengl.com/ bajo la Licencia CC BY 4.0.

De manera similar, cualquier parámetro de la fórmula de iluminación puede ser modificado utilizando un mapa de textura, ya sea reemplazando un valor, multiplicándolo, sumándolo, o cambiándolo de alguna otra forma.

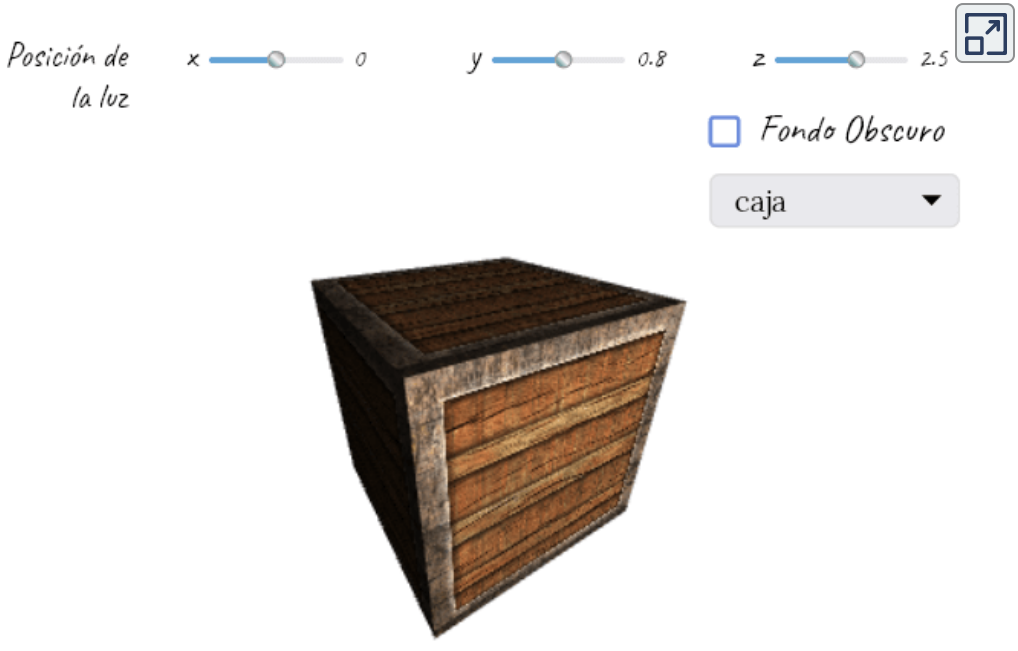


Figura 7.21. Mapeo de materiales. El lector puede mover la cámara con el ratón o dedo y alejarse o acercase con la rueda del ratón. Texturas de la caja obtenidas de https://learnopengl.com/ bajo la Licencia CC BY 4.0. Textura de la esfera obtenida de ambientCG.com.

7.4 Mapeo de normales y espacio tangente

Como mencionábamos al principio del capítulo la mayoría de las superficies en el mundo real no son totalmente lisas y pueden presentar muchos detalles. Con el mapeo de normales también conocido como *Bump mapping*, podemos lograr crear una ilusión de relieve sobre la superficie sin necesidad de agregar o modificar la geometría del objeto a una escala muy pequeña. Hasta ahora nos hemos limitado a calcular la normal de un punto por medio de una interpolación entre las normales definidas en los vértices, lo cual nos impide iluminar cualquier detalle dentro del triángulo de la malla. La técina de mapeo de normales consiste en utilizar el color de los texeles de una imagen para almacenar los vectores que se usarán para modificar la dirección de las normales de cada fragmento, cambiando así la apariencia de la superficie como se puede observar en la **Figura 7.22**.

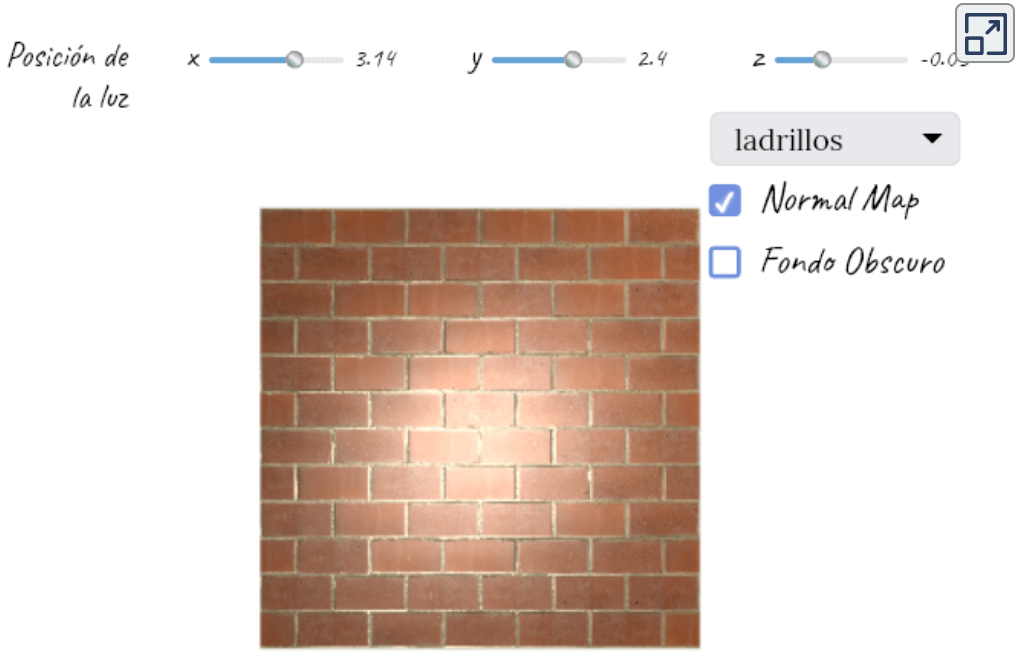


Figura 7.22. Normal mapping. Puedes dar clic en *Normal mapping* para ver la diferencia. Mueve la posición de la cámara con el ratón o dedo, o bien, puedes alejarla o acercarla con la rueda del ratón. Texturas obtenidas de ambientCG.com.

Un mapa de normales codifica los vectores normales (x, y, z) que van de $[-1, 1]$ en colores. El vector $(0, 0, 1)$ representa una normal que no es alterada, mientras que cualquier otro vector representa una modificación de la normal. Por ejemplo, si consideramos una textura de $8-$bits, entonces el 0 representaría a -1, y 255 a 1. En la **Figura 7.23** se muestra un mapa de normales, y como podemos observar el color que predomina es un color morado pastel, esto es ya que el vector inalterado $(0, 0, 1)$ es represetado con el color $(\frac{1}{2}, \frac{1}{2}, 1)$, o bien, $(128, 128, 255)$ para el ejemplo anterior. Otros colores como el rosa y verde pastel hacen referencia a las normales que están apuntando en dirección al eje x y y positivos respectivamente.

Ahora bien, si aplicamos esta misma textura sobre un plano que apunta en dirección al eje z positivo, entonces podemos simplemente extraer la normal del mapa para cada punto de la superficie y usarla directamente en la fórmula de iluminación. Sin embargo, esto dejaría de funcionar correctamente si el plano apuntara hacia una dirección diferente, ya que la mayoría de los vectores

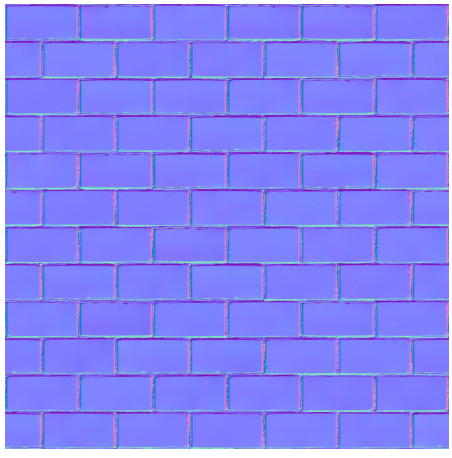


Figura 7.23. Ejemplo de Mapa de normales.

en el mapa están apuntando hacia $z+$. Este comportamiento se ilustra en la **Figura 7.24**.

Para solucionar este problema necesitamos hacer que el vector inalterado $(0, 0, 1)$ del mapa corresponda con el vector normal interpolado del fragmento, en otras palabras alinear el espacio del mapa de normales a la superficie.

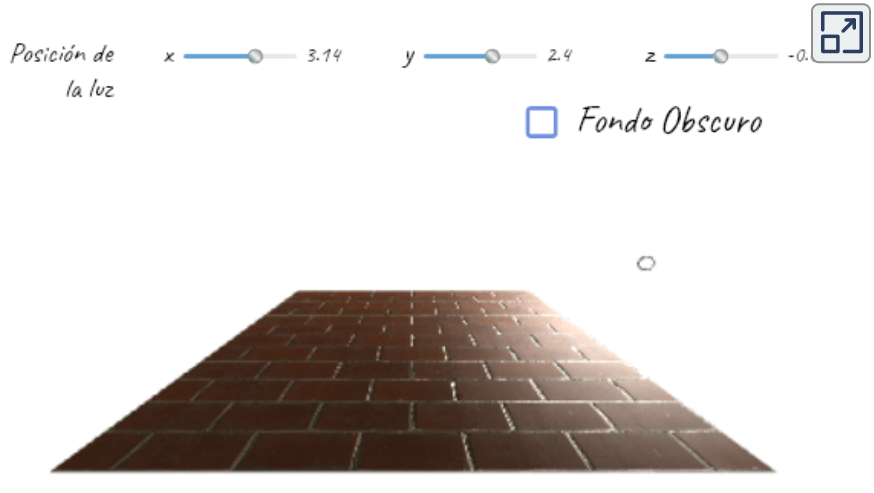


Figura 7.24. Mapeo incorrecto de normales. Puedes mover la posición de la cámara con el ratón o dedo, así como alejarla o acercarla con la rueda del ratón.

Para ello necesitamos construir un sistema de coordenadas en cada vértice de la superficie, en donde la normal del vértice siempre apunte hacia $z+$. A este sistema se le conoce como *espacio tangente* y es en donde están definidos nuestros vectores normales del mapa. La base ortogonal del espacio tangente se define con el vector normal del vértice y dos vectores tangentes a la superficie: el vector *tangente* y el vector *bitangente*.

Una vez establecido el sistema de coordenadas del espacio tangente podemos realizar la transformación de los vectores de un espacio a otro, es decir, ya sea usar las normales del mapa transformadas al espacio de vista (por fragmento), o bien, transformar la dirección del vector de la luz al espacio tangente (por vértice) para luego ser interpolado y calcular adecuadamente la intensidad de la luz utilizando la ecuación de Lambert. Vale la pena notar que el primer acercamiento es más costoso que el segundo.

Nuestra meta es entonces encontrar la matriz de cambio de base que nos permita transformar los vectores entre el espacio de vista y el tangete. El caso más intuitivo es transformar los vectores del espacio tangente al espacio de vista, ya que sabemos que el eje z del espacio tangente siempre será mapeado al vector normal del vértice. Por otra parte, queremos que su eje x corresponda a la dirección u del mapa de normales y el eje y se alinee con la dirección v del mapa.

Para encontrar los vectores tangentes de un solo vértice consideremos lo siguiente. Digamos que tenemos un triángulo P_0, P_1 y P_2, con (u_0, v_0), (u_1, v_1) y (u_2, v_2) como las coordenadas de textura de los vértices. Sea P_0 el vértice de interés tenemos que

$$\mathbf{Q}_1 = P_1 - P_0$$

$$\mathbf{Q}_2 = P_2 - P_0$$

entonces necesitamos resolver

$$\mathbf{Q}_1 = (u_1 - u_0)\mathbf{T} + (v_1 - v_0)\mathbf{B} = \Delta u_1 \mathbf{T} + \Delta v_1 \mathbf{B}$$

$$\mathbf{Q}_2 = (u_2 - u_1)\mathbf{T} + (v_2 - v_1)\mathbf{B} = \Delta u_2 \mathbf{T} + \Delta v_2 \mathbf{B}$$

donde $\mathbf{T}$ es el vector tangente y $\mathbf{B}$ el vector bitangente, los cuales están alineados con el mapa de textura, véase **Figura 7.25**.

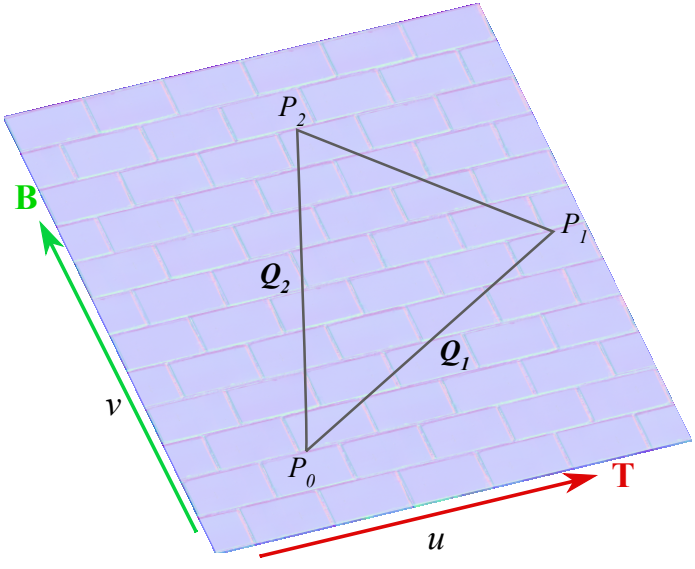


Figura 7.25. Espacio tangente.

Podemos reescribir este sistema de ecuaciones de la siguiente forma

$$\begin{bmatrix} (\mathbf{Q}_1)_x & (\mathbf{Q}_1)_y & (\mathbf{Q}_1)_z \\ (\mathbf{Q}_2)_x & (\mathbf{Q}_2)_y & (\mathbf{Q}_2)_z \end{bmatrix} = \begin{bmatrix} \Delta u_1 & \Delta v_1 \\ \Delta u_2 & \Delta v_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

Multiplicando ambas partes por la inversa de la matriz de las coordendades de textura tenemos

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix} = \frac{1}{u_1 v_2 - u_2 v_1} \begin{bmatrix} \Delta v_2 & -\Delta v_1 \\ -\Delta u_2 & \Delta u_1 \end{bmatrix} \begin{bmatrix} (\mathbf{Q}_1)_x & (\mathbf{Q}_1)_y & (\mathbf{Q}_1)_z \\ (\mathbf{Q}_2)_x & (\mathbf{Q}_2)_y & (\mathbf{Q}_2)_z \end{bmatrix}$$

Obteniendo así a los vectores (no normalizados) $\mathbf{T}$ y $\mathbf{B}$ para el triángulo definido por P_0, P_1 y P_2, con los cuales podemos construir la matriz de cambio que nos permite transformar nuestros vectores del espacio tangente al espacio de vista.

$$\mathbf{v}_{vista} = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Ahora, para transformar en sentido contrario, es decir, del espacio de vista al espacio tangente, podemos tomar la inversa de la matriz. Para simplifcar los cálculos podemos ortogonalizar la matriz con el algoritmo de Gram-Schmidt y obtener así la inversa de la matriz con su transpuesta. Esto es obtener los nuevos vectores $\mathbf{T'}$ y $\mathbf{B'}$ como sigue

$$\mathbf{T'} = \mathbf{T} - (\mathbf{N} \cdot \mathbf{T})\mathbf{N}$$

$$\mathbf{B'} = \mathbf{B} - (\mathbf{N} \cdot \mathbf{B})\mathbf{N} - (\mathbf{T'} \cdot \mathbf{B})\mathbf{T'}$$

Finalmente los vectores serian normalizados y almacenados en los vértices para construir la matriz

$$\mathbf{v}_{tangente} = \begin{bmatrix} T'_x & T'_y & T'_z \\ B'_x & B'_y & B'_z \\ N_x & N_y & N_z \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Otra optimización sería calcular el vector bitangente con el pruducto cruz

$$B' = m(\mathbf{N} \times \mathbf{T'})$$

donde $m = \pm 1$ es la dirección del vector en el espacio tangente y es igual al determinante de la matriz. De esta manera simplemente se guardaría en cada vértice el valor de m en lugar del vector $\mathbf{B'}$.

7.5 Mapeo de desplazamiento

Un problema con el mapeo de normales es que algunos detalles como elevaciones o bultos no proyectan una sombra y la silueta no se ve afectada por las irregularidades que la superficie debería presentar. Esto se debe a que realmente no estamos modificando la geometría del objeto. El mapeo de desplazamiento por otra parte consiste en modficiar la geometría usando una textura, siendo más específicos, modifica la posición de los vértices de acuerdo a los valores de desplazamiento almacenados en el mapa de textura llamado mapa de desplazamiento. Dicho mapa puede ser una imagen en escala de grises ya que solo nos interesa extraer un solo valor, o bien, una textura procedural.

Para cada vértice se realiza un desplazamiento en dirección de su normal, esto es

$$\mathbf{v} = \mathbf{v} + d(u, v) \cdot \mathbf{n}$$

donde d es la función que obtiene el valor de desplazamiento del mapa dadas las coordenadas de textura del vértice.

De manera similar, podemos aplicar el desplazamiento para cada fragmento antes de evaluar la ecuación de iluminación, generando la ilusión de una superficie rugosa, sin necesidad de agregar más geometría.

Desafortunadamente, esta técnica requiere de un mayor número de vértices para lograr que la superficie se vea más realista, por lo que necesitará de más procesamiento y memoria.

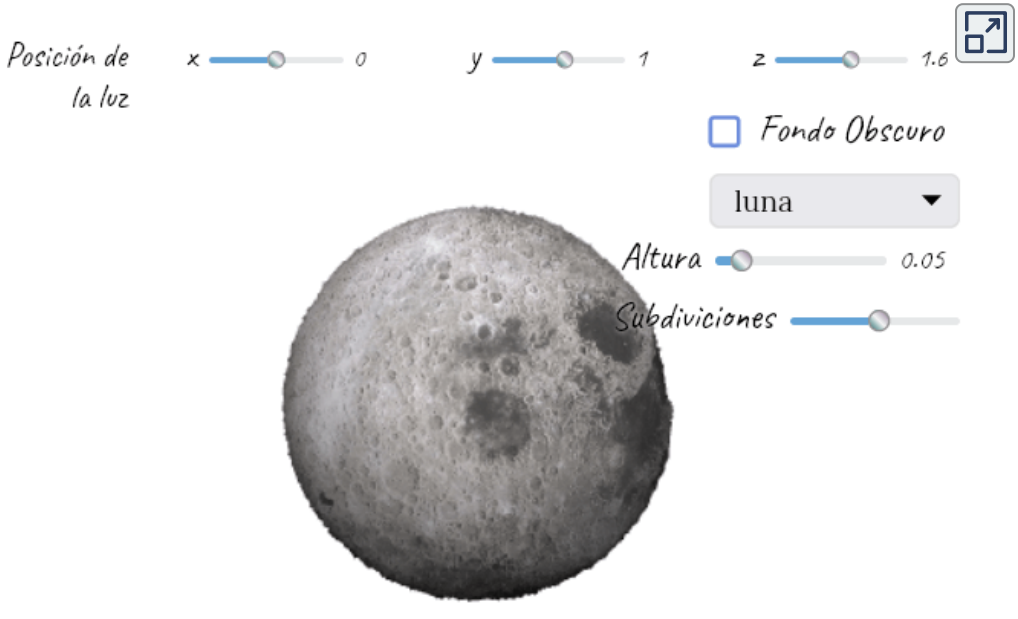


Figura 7.26. Mapeo de desplazamiento. Modifica el valor de la altura y el número de subdivisiones de la esfera, nótese que la definición de las elevaciones aumenta con un mayor número de subdivisiones. El lector puede mover la posición de la cámara con el ratón o dedo, así como también alejarla o acercarla con la rueda del ratón. Textura de la luna obtenida de https://www.solarsystemscope.com/s bajo la Licencia CC BY 4.0.

7.6 Texturas procedurales

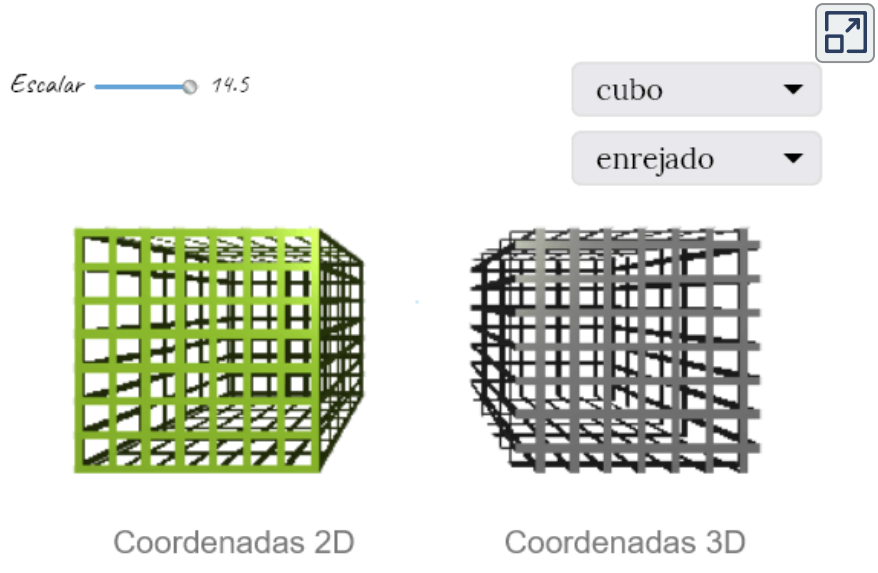


Figura 7.27. Ejemplos de texturas procedurales geométricas básicas. Se muestran distintas texturas procedurales geométricas, las cuales se muestran en función de las coordenadas de textura 2D (uv) asignadas a cada vértice (izquierda), y sobre las coordenadas 3D de la superficie (derecha). Puedes elegir la textura a utilizar, así como el objeto. Modifica el valor *Escalar* para aplicar una transformación de escalamiento sobre las coordenadas y observa las diferencias. En algunos casos, las características visuales de la textura tienen un aspecto parecido debido a que las coordenadas de textura y las del objeto tienen más o menos el mismo tamaño, por lo que es necesario aumentar el factor de escalamiento. Las coordenadas del cubo se encuentran dentro del rango $[-.5, .5]^3$ mientras que las de la tétera estan entre $[-1, 1]^3$. Se puede mover la posición de la cámara con el ratón o dedo, así como alejarla o acercarla con la rueda del ratón.

Una textura procedural es una función o algoritmo que devuelve un color para cada punto de la superficie de un objeto. Dicha función puede depender de sus coordenadas de textura (u, v), o bien de sus coordenadas en el espacio (x, y, z), a esta última se les conoce como *texturas de volumen* o 3D.

Una de las ventajas que tienen las texturas procedurales sobre las imágenes es que pueden ser renderizadas en cualquier resolución, por lo que no presentará los problemas que suelen aparecer cuando se renderiza una imagen de cerca. Sin embargo, aún pueden presentar efectos de *aliasing* .

Otra ventaja es que no se necesita de memoria extra para los mapas, ya que los valores son calculados en tiempo real.

Por otra parte, si el algoritmo es bastante complejo podría tomar mucho más tiempo que simplemente muestrear una imagen, además de que podría ser mucho más complicado crear un algoritmo que describa lo que queremos dibujar, o bien, no estar a nuestro alcance.

Si bien, los colores calculados por la función procedural suelen ser utilizados para reemplzar el color difuso de la superficie, también pueden ser utilizdos como mapas de texturas de las dos técinas anteriores, permitiéndonos modificar la dirección de las normales, o bien, la geometría de la superficie, otro aspecto que también podría modificarse es la transparencia de la misma. De hecho, también podría realizarse una combinación de éstas técinas de mapeado, incluyendo el de imágenes, para crear así un aspecto mucho más realista.

Las texturas procedurales son excelentes para simular superficies naturales como la madera, el mármol, granito, nubes, corteza de árbol, fractales, etc. Podemos dividirlas en dos tipos: *geométricas* y *aleatorias*.

Las texturas procedurales geométricas son aquellas que como su nombre lo indica generan patrones geométricos, algunos ejemplos son líneas, puntos, triángulos, cículos, estrellas, patrones hexágonales (panal de abejas), el patrón de un tablero de ajedrez, una pared de ladrillos, fractales, entre muchas otras. En la **Figura 7.27** se muestran algunos ejemplos básicos de este tipo.

Mientras que las texturas procedurales aleatorias son aquellas que requieren de una función pseudo-aleatoria, la cual suele estar correlacionada espacialmente, es decir que si dos puntos se encuentran cerca entonces compartiran parte de sus propiedaes, pero si se encuentran muy separados entonces sus propiedades calculadas serán muy independientes entre sí. Una de las funciones más utilizadas es la función de ruido de Perlín, la cual permite simular nubes, explosiones, humo, fuego, entre muchos otros efectos, en la **Figura 7.28** se muestra ejemplos de esta función. Una variación de esta función, es la función de turbulencia.

Otra función es la textura celular, con la cual se crean patrones que parecen células, losas, piel de lagarto, entre otras, del mismo modo en la **Figura 7.28** se ejemplifica el ruido de Worley que una función de este tipo. Otro ejemplo sería el resultado de una simulación física o de algún otro proceso interactivo, como las ondas de agua o la propagación de grietas.

Por otro lado, la combinación de ambos tipos de funciones puede producir un mejor resultado. Un ejemplo sería un patron de ladrillos o losas irregulares.

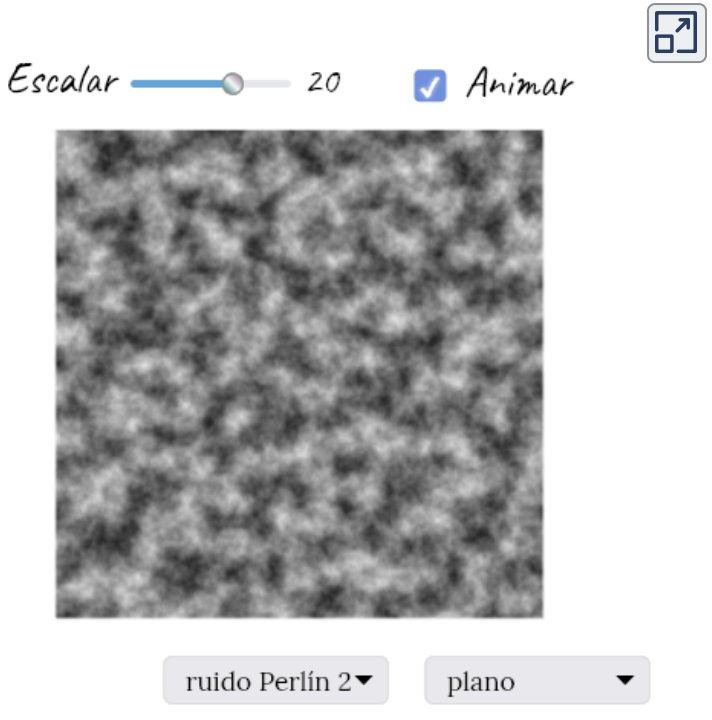


Figura 7.28. Ejemplos de texturas procedurales aleatorias. Elige el tipo de función a utilizar, así como la figura sobre la cual quieres que se proyecte. Cambia el valor *Escalar* de las coordenadas de textura pausando la *Animación*. También puesde mover la posición de la cámara con el ratón o dedo, así como alejarla o acercarla con la rueda del ratón.

Te invitamos a revisar la página de Shadertoy para ver increíbles ejemplos de texturas procedurales.

7.7 Mapeo ambiental (*environment mapping*)

El mapeo ambiental más conocido como *environment mapping* es el conjunto de técinas que nos permiten simular el efecto de reflexión del entorno sobre un objeto brillante en tiempo real, así como también simular el efecto de refracción de un objeto translucido o transparente. En el mapeo ambiental el objeto es rodeado por una superficie tridimensional cerrada sobre la cual es proyectada el entorno, el cual es almacenado en una textura o conjunto de texturas llamadas como mapa de entorno o *environment map*.

El caso más básico de enviroment mapping es el *reflection mapping*, en donde la superficie refleja su entorno como un espejo, consiguiendo una apariencia cromada. De manera similar a la reflexión especular, ésta depende de la posición del observador, solo que en este caso se calcula el vector de reflexión $\mathbf{R}$ entre el vector de vista $\mathbf{V}$ y la normal $\mathbf{N}$, esto es

$$\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{V})\mathbf{N} - \mathbf{V}$$

una vez calculado este vector se procede a calcular las coordenadas de textura del mapa de entorno y se obtiene el color. De esta manera las coordenadas de textura cambiaran cuando el observador se mueva, haciendo que parezca que el objeto está reflejando su entorno.

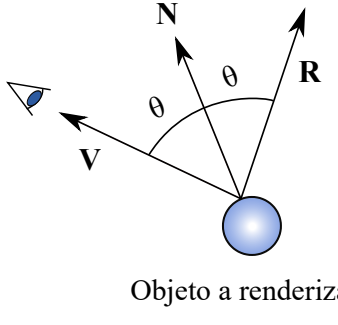


Figura 7.29. Vector de reflexión entre el vector normal y de vista.

El mapa de entorno suele ser una textura a la cual se accede utilizando coordenadas esféricas, o bien, un conjunto de 6 texturas. Nosotros nos enfocaremos en el segundo caso.

En 1986, Green introdujo el *Cube mapping*, este método es el más utilizado hoy en día, y su proyección está implementada en los GPUs modernos. Las texturas de mapas de cubo o *cube maps* están compuestos por 6 imágenes cuadradas, cada una asociada a una cara del cubo y que en conjunto forman un entorno, éstas suelen ser visualizadas con un diagrama de cruz (**Figura 7.30**).



Figura 7.30. Ejemplo de environment map de Green. Cube map obtenido de http://www.humus.name/index.php?page=Textures bajo la Licencia CC BY 3.0.

Para acceder a la textura, se toman las coordenadas de los vectores que especifican la dirección de un rayo que van desde el centro del cubo hacia fuera del cubo. La coordenada de mayor magnitud del vector selecciona la cara del mapa, luego para obtener las coordenadas (u, v) se dividen las otras dos coordenadas con el valor absoluto de la coordenada de mayor magnitud, y como ahora se encuentran dentro del rango $[-1, 1]$ solo falta mapearlas dentro del rango $[0, 1]$. Por ejemplo, sea $\mathbf{b}$ nuestro vector, el cual apunta en dirección a la cara derecha, entonces su coordenada x es la coordenada de mayor magnitud. En este caso calculamos

$$(u, v) = \left(\frac{z + x}{2|x|}, \frac{y + x}{2|x|} \right)$$

esto se resuelve de manera análoga para las demás caras.

De manera similar, realizamos este mismo cálculo para cualquier dirección de reflección $\mathbf{R}$, véase **Figura 7.31**.

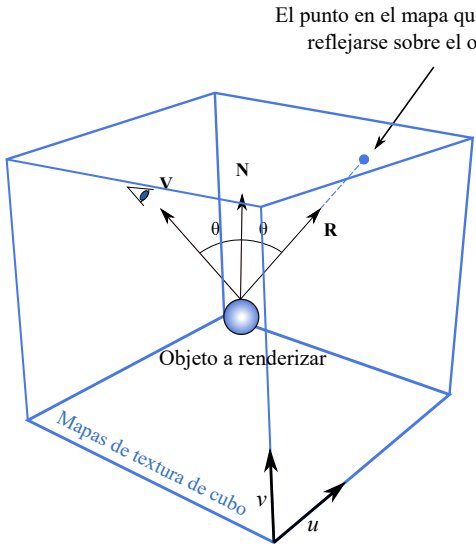


Figura 7.31. *Cube mapping*. El observador **V** ve el objeto y el vector de reflección **R** es calculado con **V** y **N**. Con el vector de reflexión se accede al mapa ambiental, convirtiendo sus coordenadas a coordenadas de textura.

Comúnmente también queremos observar el entorno que es reflejado sobre el objeto, para ello simplemente se dibuja un gran cubo usando los mapas del entorno con el objeto dentro de éste. A esta técnica se le conoce como *Skybox* y es ampliamente utilizada en videojuegos. El skybox nos permite representar el entorno de fondo a un bajo costo, y por lo general contiende objetos distantes al observador como el sol, montañas y nubes, aunque también puede describir escenas más pequeñas como la de una habitación. Las coordenadas de textura son obtenidas directamente por las coordenadas del cubo, es decir calculamos a **b** con los vértices de cada cara y los vectores interpolados entre éstas.

Ahora bien, consideremos los siguientes dos casos. El primero es cuando queremos mover los objetos dentro del skybox, para esto el skybox se mantendría estático, por lo que no tendremos ningún problema obteniendo los valores del mapa de la manera que ya hemos visto. El segundo caso es cuando queremos mover la posición de la cámara, entonces la transformación de vista tendría que ser aplicada tanto al objeto con cube mapping como al skybox, sin embargo, el que el skybox se mueva no quiere decir que la textura lo haga también, es decir, el objeto estará extrayendo los valores del mapa en su posición original.

Por ejemplo si rotamos -40 grados la vista, como se muestra en la **Figura 7.32**, entonces obtendremos al punto rosa sobre el mapa de cubo original (el cubo azul), y lo que nos interesa es obtener el punto azul de la textura. Por lo que necesitaríamos simplemente rotar al vector de $\mathbf{R}$ de regreso para obtener al punto correcto en la textura. Esto es aplicar una transformación inversa de la transformación de vista.



Figura 7.32. Correción del vector de reflexión.

Cabe aclarar que el cube mapping no nos permite reflejar los otros objetos en la escena, ya que se trata siemplemente de una manera de mapear las texturas del cubo sobre el objeto. El objeto a renderizar también puede combinar esta técina junto alguna otra de las técnica de mapeado para crear superficies que no sean totalmente reflexivas.

Si es de tu interés puedes revisar esta página para conocer un poco de la historia del environment mapping.

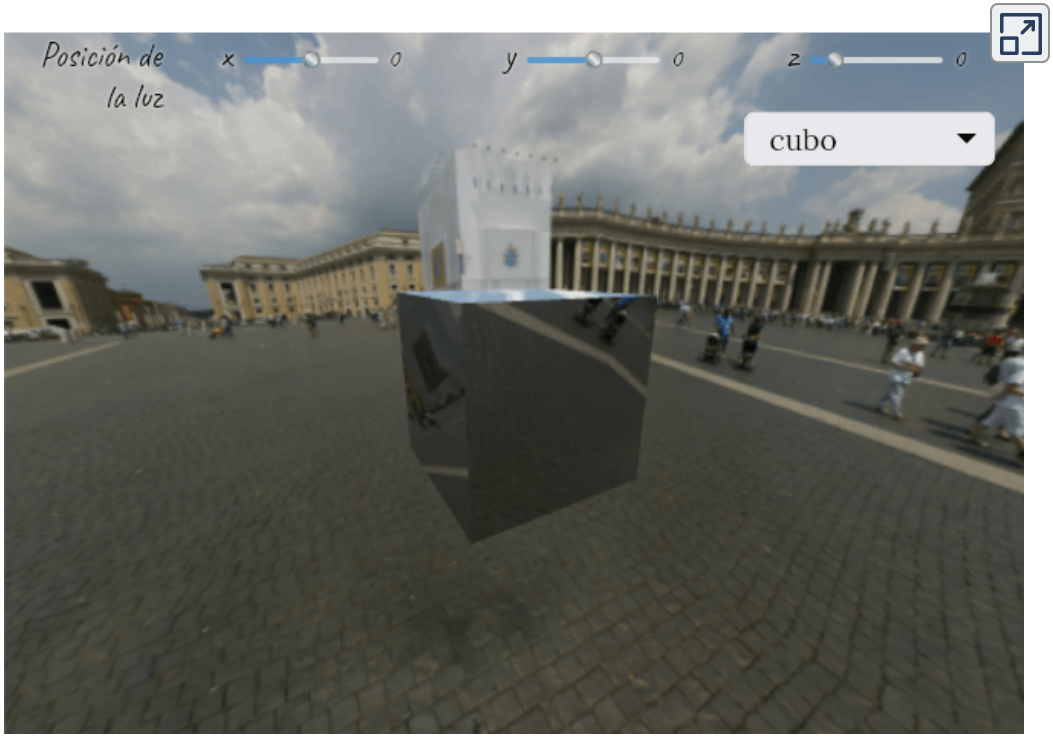


Figura 7.33. Cube mapping. Mueve la posición de la cámara con el ratón o dedo y observa cómo cambia la parte del entorno que es reflejada sobre el objeto de acuerdo a su posición. Texturas de una parte de la plaza frente a la Basílica de San Pedro obtenidas de http://www.humus.name/index.php?page=Textures.

7.8 Referencias bibliográficas

[122] Akenine-Möller, Tomas, et al. ***Real-Time Rendering.*** A K Peters/CRC Press. 4ª Edición. 2018.

[123] Ganovelli, Fabio, et al. ***Introduction to Computer Graphics a Practical Learning Approach.*** CRC Press. 2015.

[124] Shirley, Peter, et al. ***Fundamentals of Computer Graphics.*** A K Press. 3ª Edición. 2009.

[125] Lengye, Eric. ***Mathematics for 3D Game Programming and Computer Graphics.*** Course Technology, a part of Cengage Learning. 3ª Edición. 2012.

[126] Ribelles, José, et al. ***Informática Gráfica.*** Publicacions de la Universitat Jaume I. 2ª Edición. 2019.

[127] Eck, David. ***Introduction to Computer Graphics.*** Version 1.2. 2018. http://math.hws.edu/graphicsbook

[128] Eck, David. ***Generated Textures Coordinates***. Live Demos. https://math.hws.edu/graphicsbook/demos/c7/generated-texcoords.html

[129] Eck, David. ***Textures and Texture Transformations***. Live Demos. https://math.hws.edu/eck/cs424/graphicsbook2018/demos/c4/texture-transform.html

[130] Eck, David. ***2D and 3D Procedural Textures in WebGL***. Live Demos. https://math.hws.edu/graphicsbook/demos/c7/procedural-textures.html

[131] Eck, David. ***WebGL Reflection Map With Skybox***. Live Demos. https://math.hws.edu/eck/cs424/graphicsbook2018/source/webgl/skybox-and-env-map.html

[132] de Vries, Joey. 2014. ***Lighting maps***. LearnOpenGL. https://learnopengl.com/Lighting/Lighting-maps

[133] de Vries, Joey. 2014. ***Blending***. LearnOpenGL. https://learnopengl.com/Advanced-OpenGL/Blending

[134] Brown, C. Wayne. s.f. **11.9 - *Heightmaps / Displacement Maps***. Learn Computer Graphics using WebGL. Recuperado el día 14 de Junio del 2021 de https://csawesome.runestone.academy/runestone/books/published/learnwebgl2/11_surface_properties/09_heightmaps.html

[135] de Vries, Joey. 2014. ***Cubemaps***. LearnOpenGL. https://learnopengl.com/Advanced-OpenGL/Cubemaps

[136] ***Cube mapping***. 10 de Diciembre del 2020. Wikipedia. https://en.wikipedia.org/w/index.php?title=Cube_mapping&direction=prev&oldid=1042079006

[137] Kromster. 1 de Febrero del 2011. ***Re: Should I use textures not sized to a power of 2?***. [Comentario en el foro *Should I use textures not sized to a power of 2?*] Stackexchange. https://gamedev.stackexchange.com/questions/7927/should-i-use-textures-not-sized-to-a-power-of-2


DESCARTES
matemáticas interactivas