

Tema 1 - Introducción.

1.1 Concepto de entorno virtual.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Contenido del tema

Tema 1: Introducción.

- 1.1 Concepto de entorno virtual.
- 1.2 Percepción y sentidos. Visualización 3D.
- 1.3 Modelos 3D. Modelos volumétricos, sólidos y de superficie.
- 1.4 Sistemas de interacción 2D y 3D.

1.1 Concepto de entorno virtual.

¿Entornos virtuales?

- Realidad Virtual
- Realidad Aumentada
- Realidad Mixta
- Gráficos interactivos
- Video 360°
- Visualización

Realidad Virtual.

La **Realidad Virtual** (RV) introduce al usuario en un mundo virtual a través de sus sentidos.

- **Inmersión:** «Puentear» los sentidos para introducir a la persona en un entorno virtual.
- 5 sentidos: Sistemas especiales para puentear todos estos sentidos.



Figura 1: Entorno de realidad virtual.

Realidad Aumentada.

La **Realidad Aumentada** (RA) combina imágenes reales capturadas mediante cámaras con modelos 3D ubicados en ese entorno.

- **Expansión:** Permite añadir información visual en entornos reales.
- **Tracking:** Necesitamos ubicar la posición y orientación en tiempo real y «casarla» con el mundo virtual.



Figura 2: Entorno de realidad aumentada.

Realidad Mixta.

- Combina RV y RA en distintos grados.

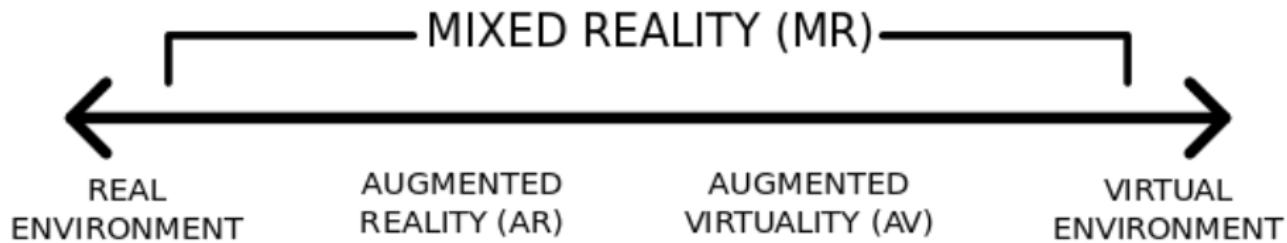


Figura 3: Espectro de la realidad mixta.

Gráficos interactivos.

- Gráficos 2D + 3D en una pantalla: permite interaccionar con interfaces 2D (ratón, pantallas táctiles, etc.).



Figura 4: Ejemplo de gráficos interactivos (demo Unreal).

Vídeos 360°.

- Permite cambiar la dirección de la vista, vídeo real.

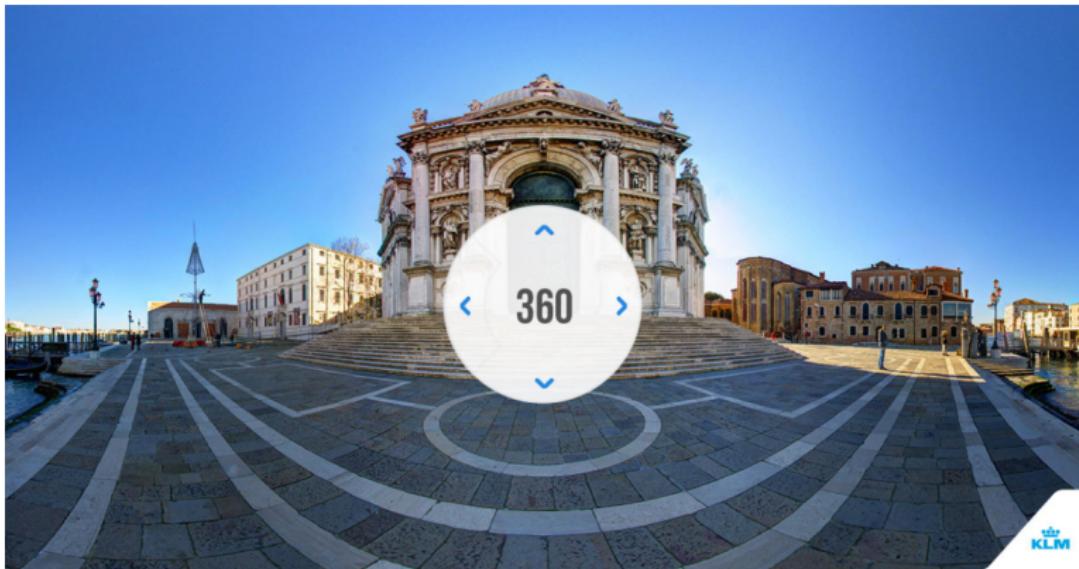


Figura 5: Ejemplo de video 360°.

Renderización.

Renderización: Proceso por el que se puede generar imágenes 2D a partir de modelos 3D.

- **Fotorrealismo:** busca el imitar los procesos físicos de la luz y materiales.
 - ▶ **Objetivo:** resultados indistinguibles de fotografías.
- **Visualización expresiva (Non-photorealistic rendering):** busca asemejarse a dibujos artísticos (rallado, tinta, punteado, pintura, etc.).
 - ▶ **Objetivo:** pasar por ser una obra de arte.

Prueba de Turing (I).



Figura 6: Comparación de fotografía y renderizado fotorrealista.

Prueba de Turing (II).

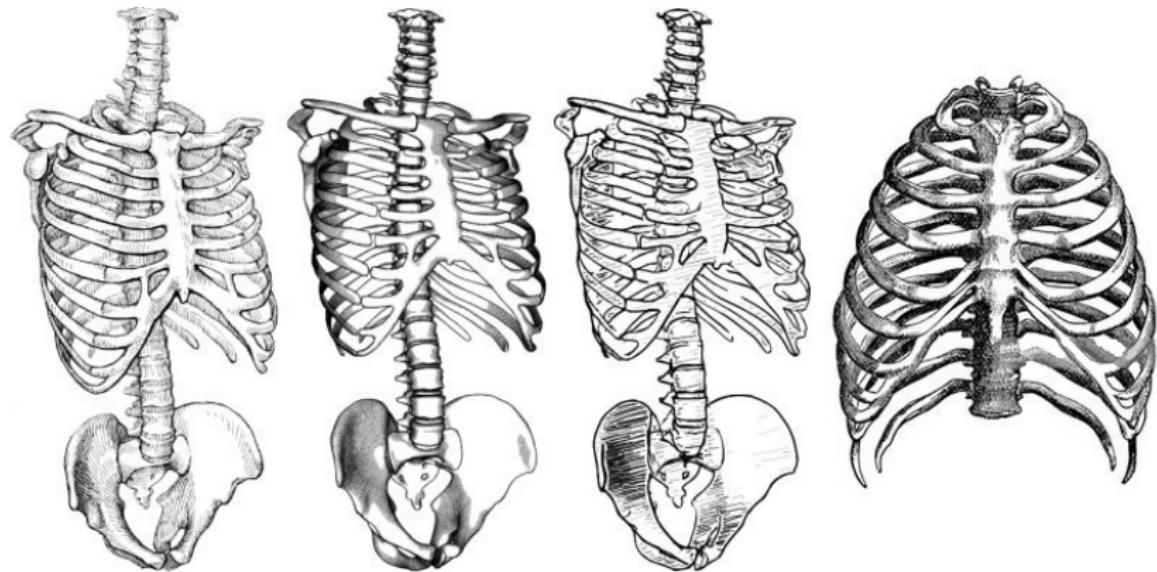


Figura 7: Comparación de visualización expresiva e ilustración.

Entorno Virtual.

Un **Entorno Virtual** (EV) se define como un entorno generado por ordenador que simula el mundo real.

[...] these environments can be completely immersive, hardware-based, three-dimensional interactive experiences utilizing sound and force feedback to simulate, as accurately as possible, a real environment. [1]

[1] S.K. Gupta, D.K. Anand, J. Brough, M. Schwartz, and R. Kavetsky: «Training in Virtual Environments. A safe, cost effective, and engaging approach to training». University of Maryland. 2008.

Tema 1 - Introducción.

1.2 Percepción y sentidos. Visualización 3D.

Germán Arroyo, Juan Carlos Torres

28 de febrero de 2021

Contenido del tema

Tema 1: Introducción.

- 1.1 Concepto de entorno virtual.
- 1.2 Percepción y sentidos. Visualización 3D.
- 1.3 Modelos 3D. Modelos volumétricos, sólidos y de superficie.
- 1.4 Sistemas de interacción 2D y 3D.

1.2 Percepción y sentidos. Visualización 3D.

Sentido del olfato:

Dispositivos de mezcla de aromas: <https://youtu.be/YxFXjKn1LxQ?t=85>

Sentido del gusto:

Dispositivos de inducción química: <https://youtu.be/YxFXjKn1LxQ?t=177>

Sentido del tacto:

Dispositivos hápticos: <https://youtu.be/0Cry0b4TOm4?t=9>

Sentido de la audición:

Audio en 3D y auriculares: <https://youtu.be/aNia7r41cCw>

Sentido de la vista:

Cascos (visión estéreo) y sensores de movimiento:

https://youtu.be/RQ4gp-_XHvk?t=341

Ejemplo de entornos virtuales.

- Podemos crear mundos y experiencias que:
- **Existen** pero suponen riesgo: simulación y entrenamiento, entretenimiento, etc.
- **No existen** alimentando nuestra creatividad pero también tratamiento de enfermedades mentales.

Algunos ejemplos:

- Tratamiento de enfermedades mentales:
<https://www.youtube.com/watch?v=5Kla3nNmMAc>
- Estimulación mental: <https://youtu.be/5Kla3nNmMAc?t=254>
- Creación artística: <https://youtu.be/DunWmzd8kXU>
- Diseño en ingeniería: <https://youtu.be/mWaQfjEJIMQ?t=49>
- Aprendizaje: <https://youtu.be/p4vO64Y27JE?t=183>
- Simulación: <https://youtu.be/cwK3MatOQFc?t=6>

Prueba de Turing

Conseguir que el comportamiento de una persona sea indistinguible del comportamiento en un entorno real.



Figura 1: Ejemplo del uso de EV para el tratamiento de fobias.

https://youtu.be/Yw0_1ZjxWCM?t=36

Tema 1 - Introducción.

1.3 Modelos 3D. Modelos volumétricos, sólidos y de superficie.

Germán Arroyo, Juan Carlos Torres

6 de marzo de 2021

Contenido del tema

Tema 1: Introducción.

- 1.1 Concepto de entorno virtual.
- 1.2 Percepción y sentidos. Visualización 3D.
- 1.3 Modelos 3D. Modelos volumétricos, sólidos y de superficie.
- 1.4 Sistemas de interacción 2D y 3D.

1.3 Modelos 3D. Modelos volumétricos, sólidos y de superficie.

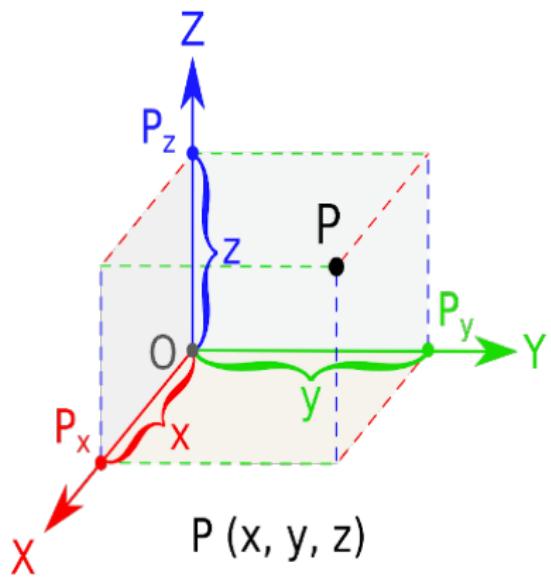


Figura 1: Espacio euclídeo 3D.

Tipos de modelos: Nube de puntos.

- **Posición de los puntos:** $P_i: (x_i, y_i, z_i) \in \mathbb{E}^3$
- **Propiedades asociadas al punto:** color, densidad, etc.



Figura 2: Nube de puntos.

Tipos de modelos: Malla.

- Solamente se representa una superficie (de forma aproximada).
- $V \subseteq \mathbb{E}^3, E \subseteq V \times V, F = \{(i_1, i_2, \dots, i_n) | i_k \in V, (i_k, i_{k+1}) \in E\}$

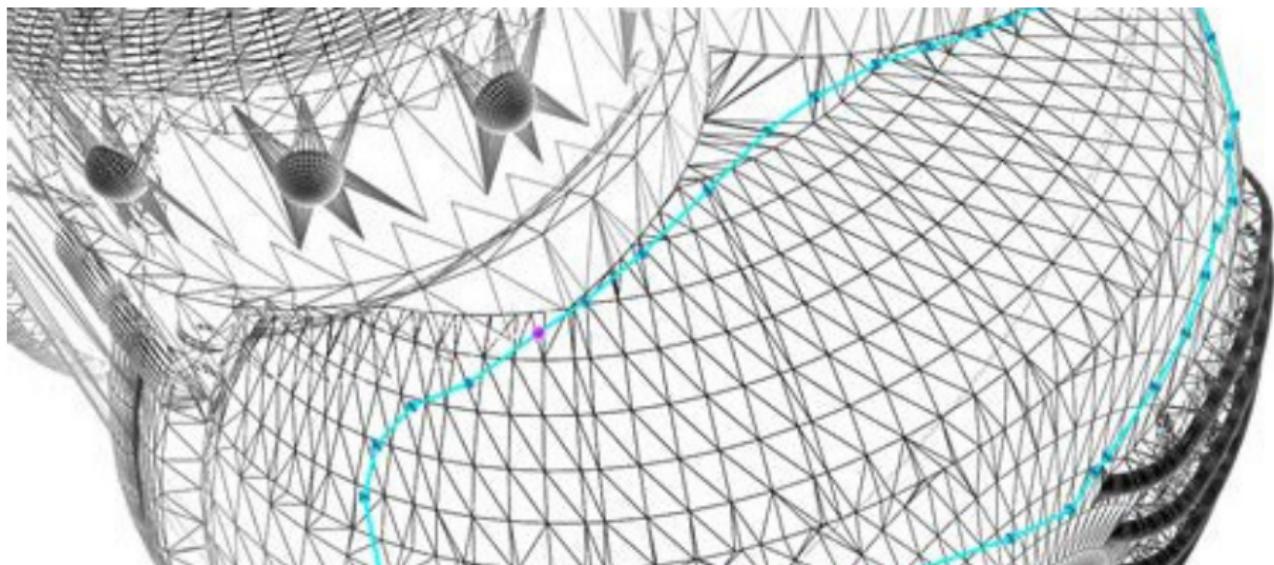


Figura 3: Modelo de malla

Tipos de modelos: Sólidos (I).

- La regularización de un conjunto se define como la clausura de su interior: $r(S) = c(i(S))$
- Un conjunto es regular si es igual a su regularización: $S = r(S)$
- Conjunto (continuo) regular cerrado: $S \subseteq \mathbb{E}^3$.

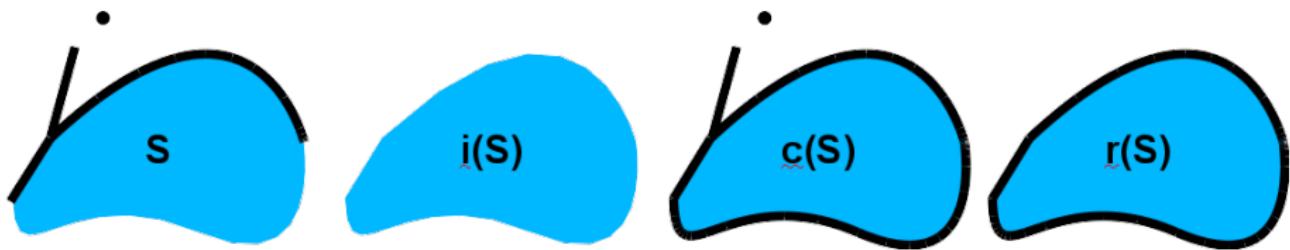
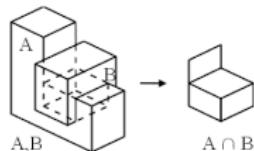


Figura 4: Modelo sólido.

Tipos de modelos: Sólidos (II).

- Todo el sólido mantiene la misma propiedad.
- Operaciones regularizadas



$$A \cap^* B = r(A \cap B) = c(i(A \cap B))$$

$$A \cup^* B = r(A \cup B) = c(i(A \cup B))$$

$$A -^* B = r(A - B) = c(i(A - B))$$

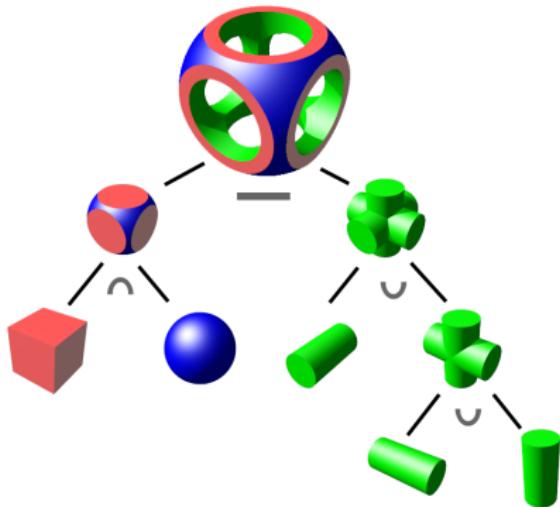
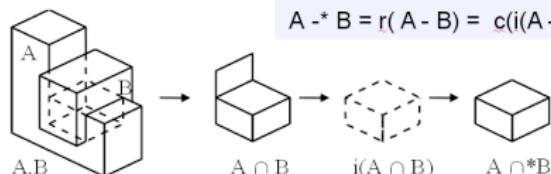


Figura 5: Modelo CSG.

Tipos de modelos: Volumétricos.

- **Rejillas de datos:** véxeles → Interior heterogéneo
- $V \subseteq \mathbb{E}^3 \times \Gamma$, donde Γ son las propiedades del véxel.

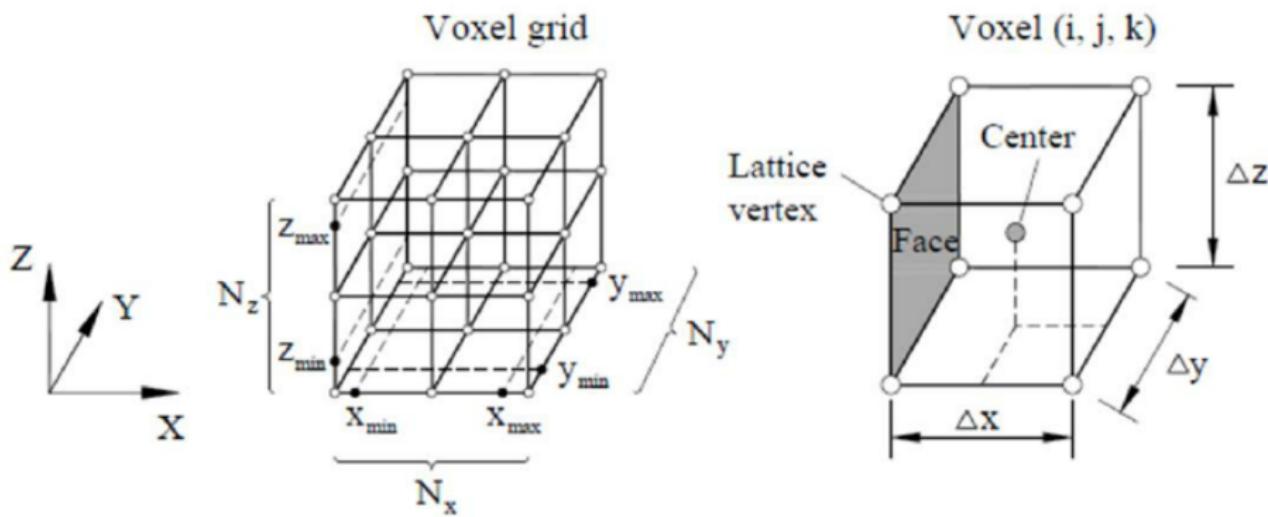


Figura 6: Rejilla de véxeles

Operaciones comunes en espacios euclídeos.

- Operaciones básicas:

- ▶ Posición (P_i): $(x_i, y_i, \dots) \in \mathbb{E}^n$
- ▶ Traslación (T): $x- > x + \Delta t_x$
- ▶ Rotación (R): $x- > x \cdot \cos(\Theta) + y \cdot \sin(\Theta)$
- ▶ Escalado (S): $x- > x \cdot \Delta s_x$

Operaciones, ¿cómo se combinan?

- $[(x, y) + (\Delta t_x)] \cdot \Delta s_x$
- ¿Es igual que esto? $[(x, y) \cdot (\Delta s_x)] + \Delta t_x$

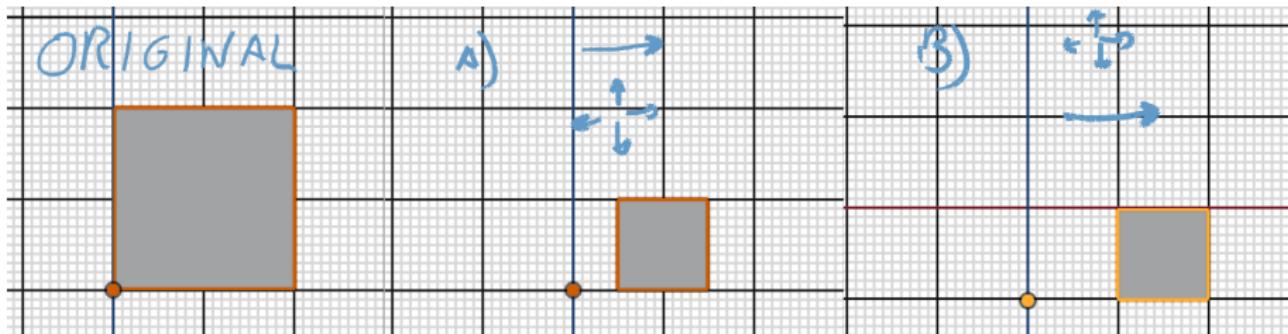


Figura 7: Operaciones geométricas y su orden.

Punto de pivote: Punto a partir del cual se realizan las transformaciones.

Notación: Matrices

- Las matrices nos permiten aprovechar uniformar todas las operaciones.

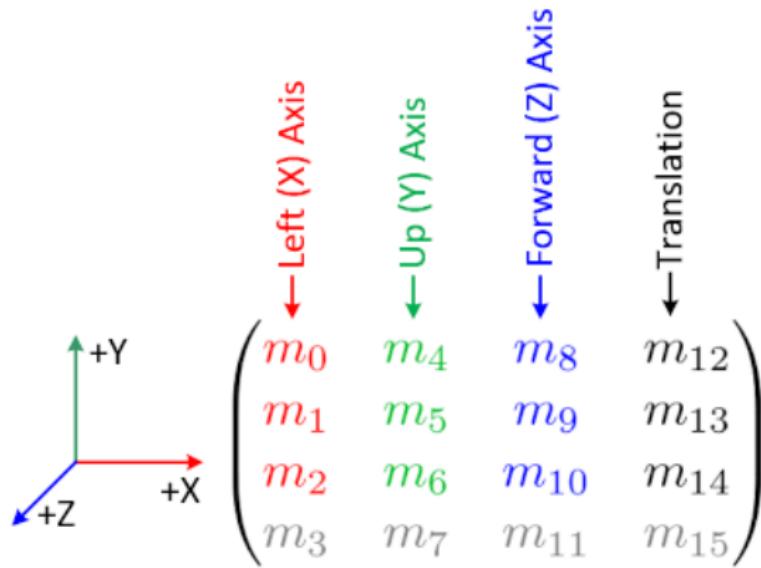


Figura 8: Ejemplo de operación representado como matriz.

Operaciones comunes: Matrices (I).

$$P_i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix}$$

$$T = \begin{pmatrix} 1 & 0 & 0 & \Delta t_x \\ 0 & 1 & 0 & \Delta t_y \\ 0 & 0 & 1 & \Delta t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow Translate(P_i) = T \cdot P_i$$

$$S = \begin{pmatrix} \Delta s_x & 0 & 0 & 0 \\ 0 & \Delta s_y & 0 & 0 \\ 0 & 0 & \Delta s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow Scale(P_i) = S \cdot P_i$$

Operaciones comunes: Matrices (II).

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \Theta_\alpha & -\sin \Theta_\alpha & 0 \\ 0 & \sin \Theta_\alpha & \cos \Theta_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos \Theta_\beta & 0 & \sin \Theta_\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \Theta_\beta & 0 & \cos \Theta_\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos \Theta_\gamma & 0 & -\sin \Theta_\gamma & 0 \\ 0 & 1 & 0 & 0 \\ \sin \Theta_\gamma & 0 & \cos \Theta_\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ventajas de las matrices.

- $R(\Theta_\alpha, \Theta_\beta, \Theta_\gamma) = R_x \cdot R_y \cdot R_z$
- $M = R(\alpha, \beta, \gamma) \cdot T \cdot S$

- $$M = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & b_1 & c_2 & c_3 \\ d_0 & b_1 & d_2 & d_3 \end{pmatrix}$$

- $M \cdot P_0$
- $M \cdot P_1$
- ...

Apilación de matrices jerárquica

- $M(P) \rightarrow M \cdot P$
- $M(N(P)) \rightarrow M \cdot N \cdot P$

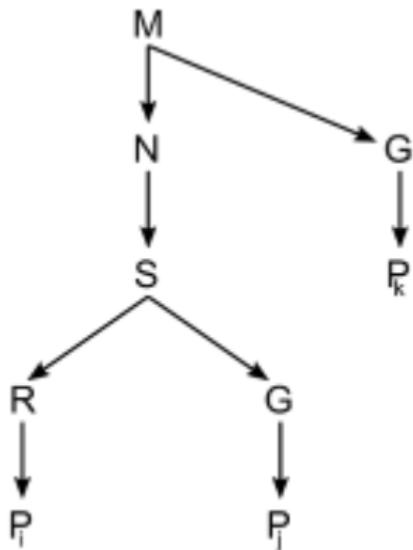


Figura 9: Diagrama de operaciones.

Tema 2 - Arquitectura y modelos para entornos virtuales.

2.1 Grafos de escena y modelos jerárquicos.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Contenido del tema

Tema 2: Arquitectura y modelos para entornos virtuales.

- 2.1 Grafos de escena y modelos jerárquicos.
- 2.2 Métodos básicos de representación.
- 2.3 Sistemas básicos de iluminación y cámaras.
- 2.4 Modelos de generación procesal.

2.1 Grafos de escena y modelos jerárquicos

Motor de grafos de escena:

- Trabajar con escenas.
- Edición y visualización 3D.
- Captura de interacción.
- Generación de cálculos y simulaciones.

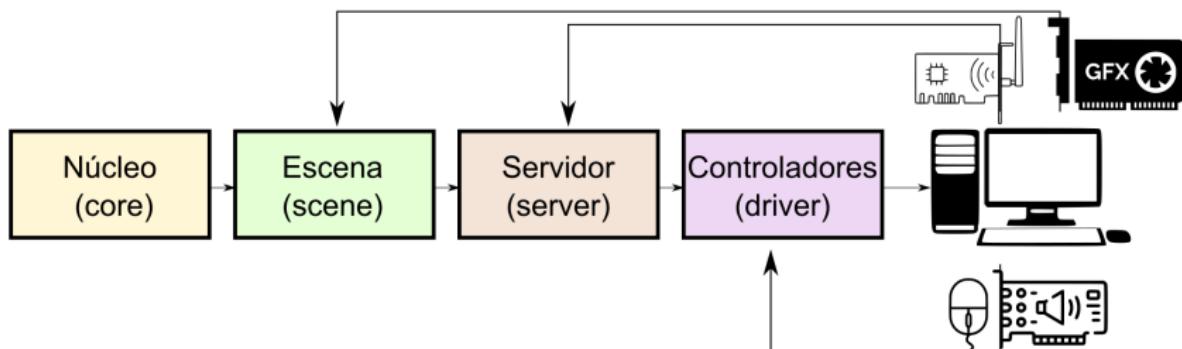


Figura 1: Esquema abstracto de motor de grafos de escena.

Escena (I)

La **Escena** contiene todos los elementos geométricos, animaciones, cálculos, etc. que serán visibles en el entorno.

Procesos de dibujado de la escena.

- Cargar recursos y transferirlos a memoria (CPU).
- Procesar/optimizar el modelo.
- Transferirlos al procesador gráfico (GPU).

Escena (II)

Una escena está definida por un grafo de escena:

- Pueden ser visualizadas o no.
- Pueden grabarse en disco y cargarse.
- Pueden ser instanciadas (una o más veces).
- Ejecutar una escena significa mostrar el mundo virtual que representa.

Recuerda.

Podemos ver la escena como un archivo ZIP que contiene una serie de elementos que conforman una porción del mundo virtual.

Grafos de escena (I)

Un **Grafo de escena** es un grafo (usualmente acíclico dirigido) que representa la estructura de una escena.

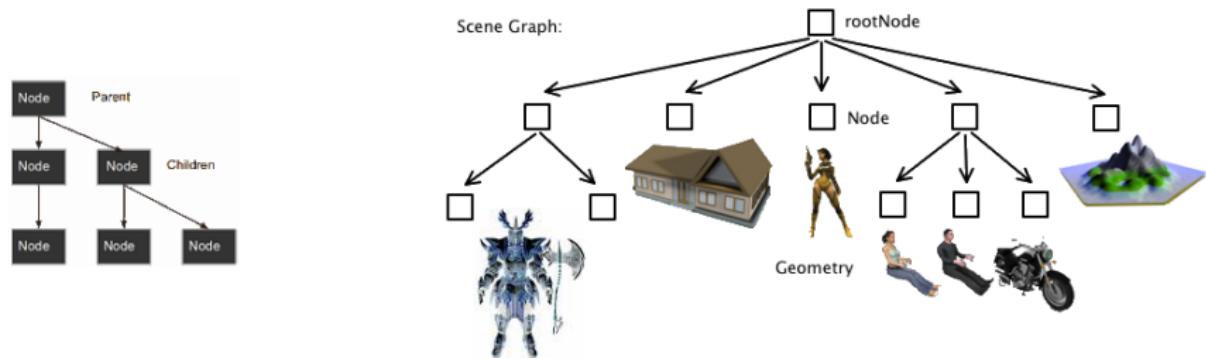


Figura 2: Representación de un grafo de escena.

Grafos de escena (II)

Los grafos de escena son útiles para:

- Entender las dependencias entre objetos.
- Diseñar el modelo geométrico.
- Editar y animar el modelo.
- Reutilizar geometría.

Nodos (I)

Un **Nodo** es la unidad básica del grafo de escena. Puede contener elementos geométricos, pero también abstractos.

- El nodo raíz representa una escena.
- Los nodos pueden contener primitivas y objetos geométricos simples (cubos, etc.), pero no está limitado a este contenido (cámaras, luces, cajas de colisión, etc.).
- Los nodos contienen propiedades (como transformaciones, color, etc.), recursos (imágenes, sonido, etc.) y funciones (código de programación).
- Una escena puede ser también un nodo (como en Godot).

Nodos (II)

Los nodos:

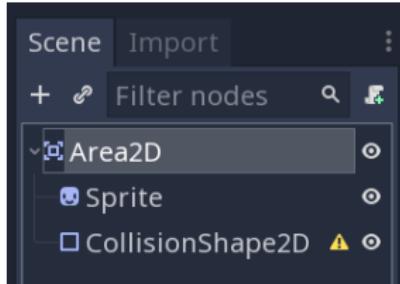
- Tienen nombre, que los identifica únicamente.
- Tiene propiedades que heredan (de una clase).
- Puede recibir eventos mediante el uso de funciones (*callbacks*).
- Pueden extenderse (para añadir funcionalidad).
- Puede tener cualquier cantidad de nodos hijos asociados.
- Siempre tiene padre, salvo el nodo raíz que representa la escena.

Jerarquía de nodos

- Un nodo padre domina las transformaciones de un nodo hijo.



- El nodo padre algunas veces afecta al comportamiento del hijo.



Tipos de nodos (I)

Los nodos suelen estar agrupados en distintas categorías.

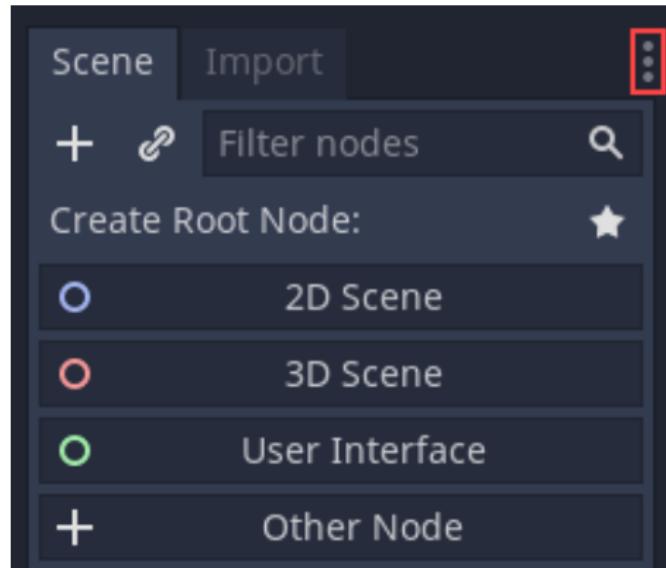


Figura 3: Tipos de nodos en Godot.

Tipos de nodos (II)

Además, cada nodo tiene una función específica en la escena que depende de la **Clase** de la que hereda.

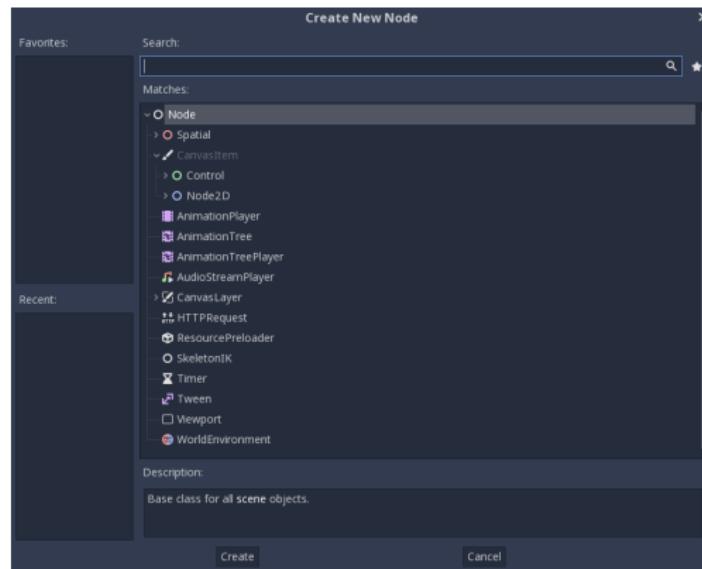


Figura 4: Algunas clases de nodos en Godot.

Propiedades de los nodos

Las propiedades de los nodos tienen tipos:

- **Sencillos:** texto, enteros, reales, etc.
- **Mapas:** texturas, imágenes, etc.
- **Referencias:** a objetos perteneciente a una clase que no llegan a ser nodos (o sí).

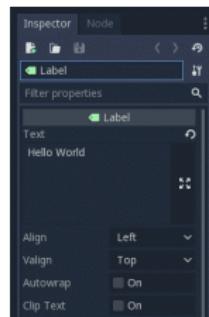


Figura 5: Propiedades de un nodo en Godot.

Instanciación de escenas (I)

Usar una escena como si fuera un nodo se llama **instanciar una escena**.

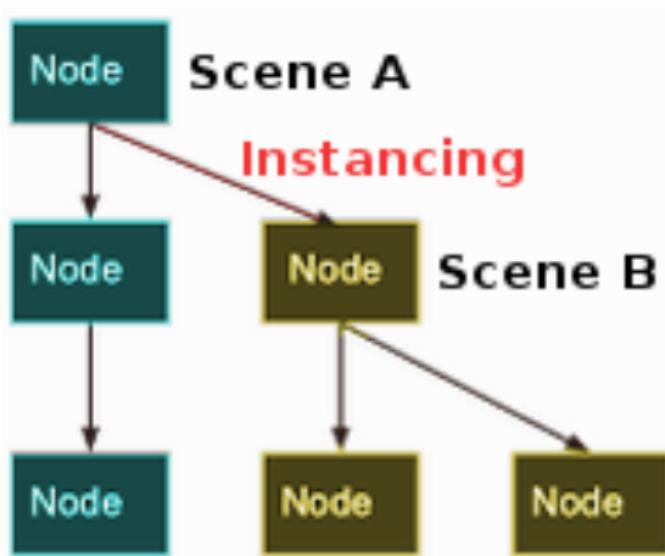


Figura 6: Instanciación de una escena.

Instanciación de escenas (II)

La instanciación de escenas permite crear objetos, comportamientos, etc. Con un comportamiento común, que luego pueden ser instanciados.

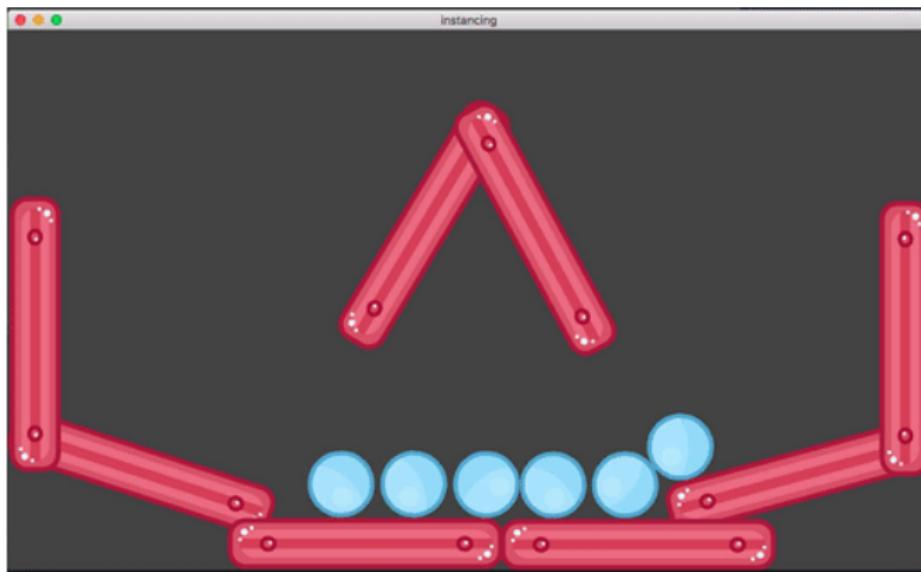


Figura 7: Ejemplo de uso de instancias.

Diseño de entornos virtuales

A la hora de diseñar entornos virtuales tenemos varias opciones:

- Modelo vista-controlador (*Model View Controller*).
- Modelo sistema de componentes de entidad (*Entity Component System*).
- **Modelo de grafo** (*Graph Model*).

Modelo vista-controlador

El patrón de diseño **Modelo vista-controlador** especifica que una aplicación consta de un modelo de datos, información de presentación e información de control.

- Cada uno de los elementos se tienen que separar en diferentes objetos.
- Fuerte relación entre la interfaz de usuario con la capa de interacción de la aplicación.

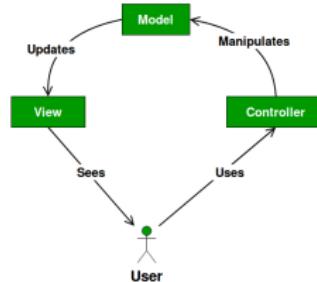


Figura 8: Esquema MVC.

Modelo sistema de componentes de entidad

- **Entidad:** la entidad es un objeto de propósito general de identidad única (ID único).
- **Componente:** los datos del objeto (desligados de la entidad) para un aspecto de la entidad (etiquetas de comportamiento).
- **Sistema:** cada sistema se ejecuta constantemente, realizando las acciones globales para cada entidad que posea un determinado tipo de componente.

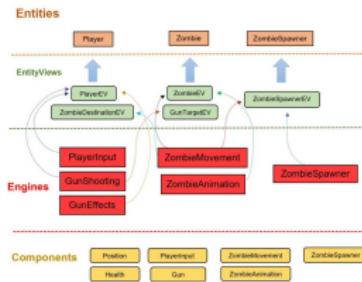
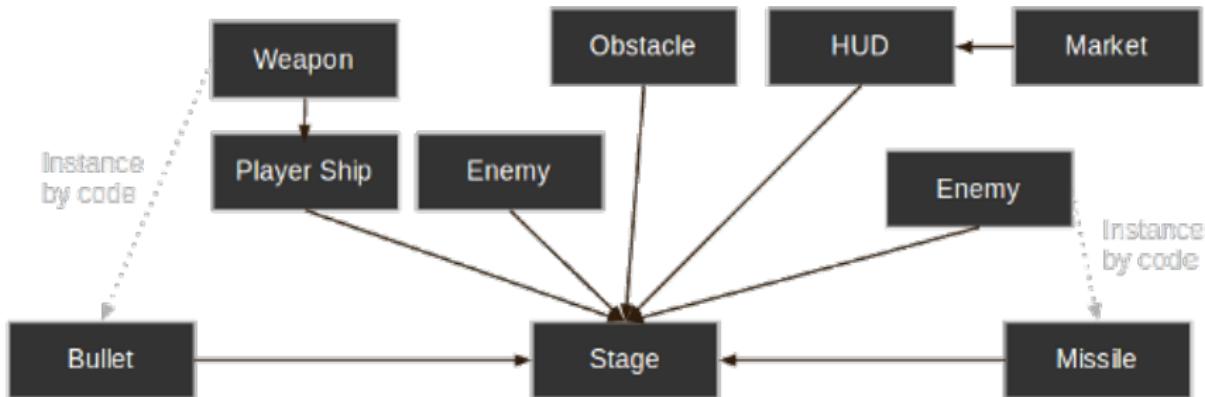


Figura 9: Esquema Entidad-Componente-Sistema.

Modelo de grafo

- Escribe las partes del mundo virtual que se visualizarán (desde un punto de vista ajeno al programador).
- Cada flecha representa pertenencia de un componente a otro.
- Las líneas punteadas indican relación mediante programación.



Diseñar usando modelos de grafo de escena

- 1 Define las escenas que habrá (habitaciones, escenarios, etc.).
- 2 Decide en que lugar empezará el usuario.
- 3 Decide muebles y objetos no interactivos.
- 4 Decide como se conectarán las escenas (que habitación conecta con otra).
- 5 Diseña los elementos interactivos y móviles.
- 6 Coloca los elementos interactivos en cada escena.

Ampliar es trivial

Si luego amplias a una casa, puedes crear una única escena que refuerce tus habitaciones. Si amplias a una ciudad, tendrás una nueva escena con instancias a la casa.

Ejercicio Teórico-Práctico:

Creación del modelo de grafo para tus prácticas.

Crea el grafo para el modelo final de la última práctica, añade al menos 3 habitaciones, sus conexiones y algunos elementos animados e interactivos.

Tema 2 - Arquitectura y modelos para entornos virtuales.

2.2 Métodos básicos de representación.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

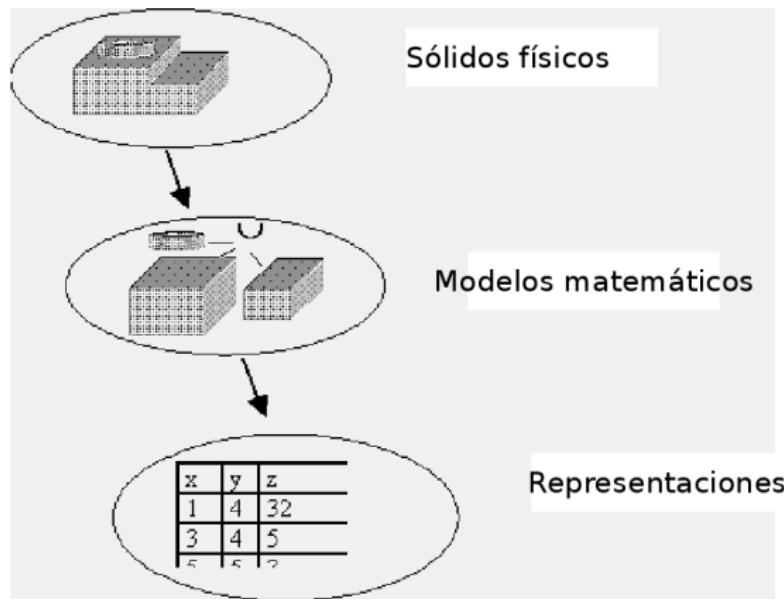
Contenido del tema

Tema 2: Arquitectura y modelos para entornos virtuales.

- 2.1 Grafos de escena y modelos jerárquicos.
- 2.2 Métodos básicos de representación.
- 2.3 Sistemas básicos de iluminación y cámaras.
- 2.4 Modelos de generación procesal.

2.1 Métodos básicos de representación

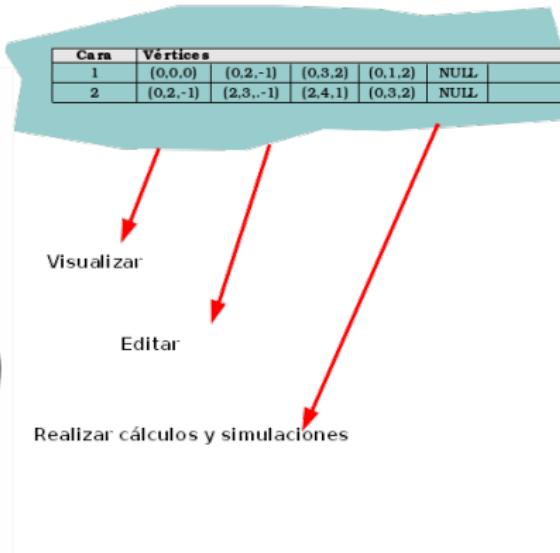
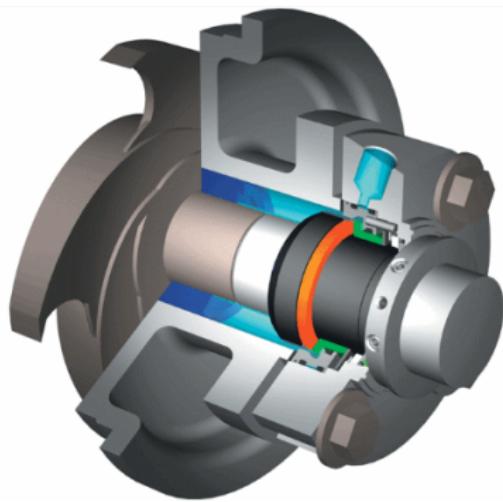
Modelos de malla que alguna vez se tratan como sólidos.



Sólidos y mallas están relacionados. ¡Pero no son lo mismo!

Sólidos

Con un sólido se pueden calcular propiedades que no se pueden calcular en una malla.



Sólidos: Modelo topológico

- Definiciones:
- **Sólido:** es un subconjunto cerrado y acotado de \mathbb{E}^3 .
- **Rígido:** dos sólidos que se diferencian tan solo en una transformación rígida (sin deformaciones) son el mismo sólido.
- **Homogeneidad tridimensional:** En el espacio 3D solamente puede haber transformaciones homogéneas (dadas por matrices 4x4).

Transformaciones homogéneas (I)

- Dada una matriz H que representa una transformación homogénea, y un vector u :

$$\blacktriangleright H = \begin{pmatrix} a_x & b_x & c_x & p_x \\ a_y & b_y & c_y & p_y \\ a_z & b_z & c_z & p_z \\ d_1 & d_2 & d_3 & 1 \end{pmatrix}; u = \begin{pmatrix} u_x \\ u_y \\ u_z \\ 1 \end{pmatrix}$$

- La transformación homogénea de u , es representada por v :
 - $v = H \cdot u$
- Es común trasponer las operaciones para calcularlas en el computador, lo que invierte el orden de cómputo:
 - $v^T = u^T \cdot H^T$

Transformaciones homogéneas (II)

- El vector $\vec{p} = (p_x, p_y, p_z, 1)$ representa una traslación.

$$\blacktriangleright T = \begin{pmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}; u = \begin{pmatrix} u_x \\ u_y \\ u_z \\ 1 \end{pmatrix}$$

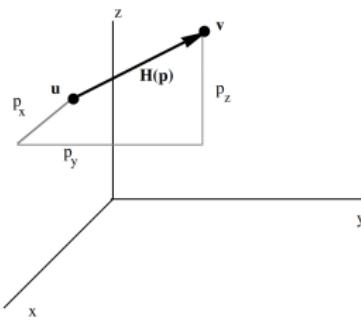


Figura 1: Traslación del punto u a v.

Transformaciones homogéneas (III)

- El vector $\vec{s} = (s_x, s_y, s_z, 1)$ representa un escalado en los tres ejes ortogonales.

$$\blacktriangleright S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; u = \begin{pmatrix} u_x \\ u_y \\ u_z \\ 1 \end{pmatrix}$$

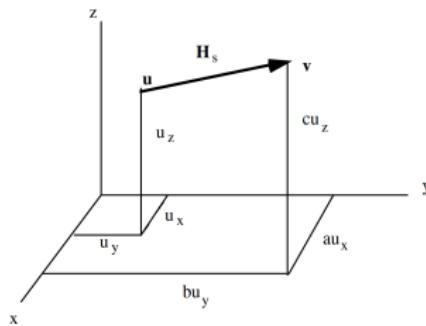
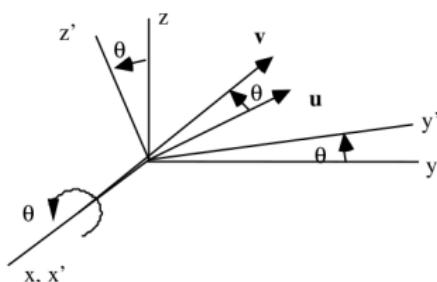


Figura 2: Escalado u a v .

Transformaciones homogéneas (VI)

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \Theta_\alpha & -\sin \Theta_\alpha & 0 \\ 0 & \sin \Theta_\alpha & \cos \Theta_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_y = \begin{pmatrix} \cos \Theta_\beta & 0 & \sin \Theta_\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \Theta_\beta & 0 & \cos \Theta_\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos \Theta_\gamma & 0 & -\sin \Theta_\gamma & 0 \\ 0 & 1 & 0 & 0 \\ \sin \Theta_\gamma & 0 & \cos \Theta_\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Transformaciones homogéneas (V)

- ¡Cuidado! Las operaciones no son commutativas, ¡tampoco las rotaciones!
 - ▶ $R_y(\pi/2) \cdot R_z(\pi/2) \neq R_z(\pi/2) \cdot R_y(\pi/2)$
- Se puede calcular la transformación inversa de cualquiera de estas matrices homogéneas (4x4):

Si: $v = R \cdot u$, $R^{-1} \cdot v = R^{-1} \cdot R \cdot u = I \cdot u = u$, **Tenemos que:**
 $u = R^{-1} \cdot v$

- ▶ En el caso de una translación y rotación, basta con invertir el signo de las magnitudes/ángulos: $t_{inv} = -t$, $\Theta_{inv} = -\Theta$.
- ▶ En el caso del escalado, hay que invertir las cantidades: $s_{inv} = s^{-1}$.

Problema con sólidos

Aunque podamos pensar que la operación entre dos elementos sólidos devuelve siempre un sólido esto no es siempre así.

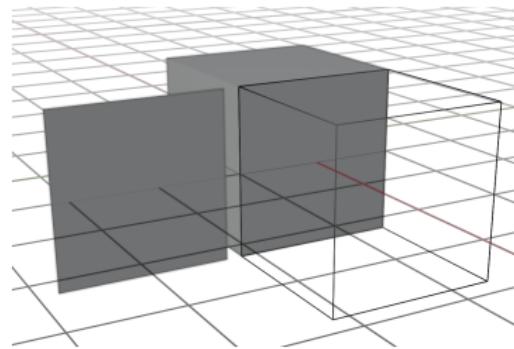


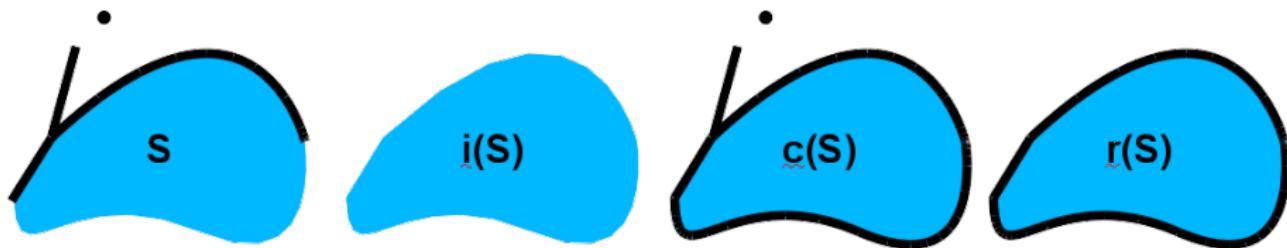
Figura 3: Ejemplo de resta entre de dos cubos.

¡Necesitamos un marco teórico para definir un sólido y evitar estos problemas!

Regularización del sólido (I)

La idea para regularizar el sólido es simple:

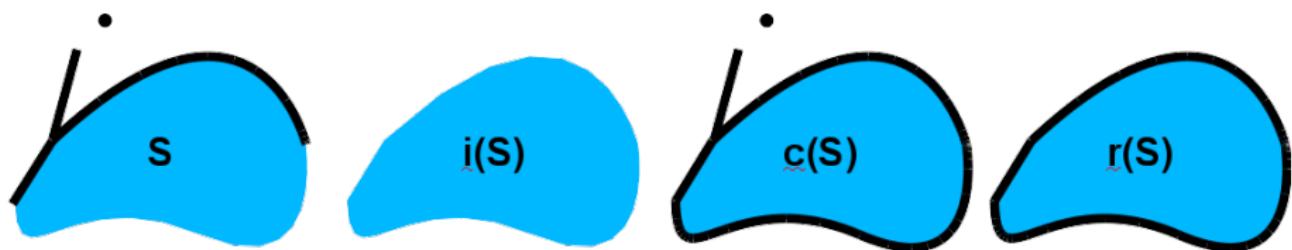
- ➊ Calcular el resultado como siempre y bajar de dimensión una vez los componentes han sido generados.
- ➋ Calcular el interior del resultado. El resultado es un sólido sin bordes (en azul).
- ➌ Calcular la clausura del resultado anterior, lo que añade los bordes (en negro).



Regularización del sólido (II)

Formalmente:

- La regularización de un conjunto se define como la clausura (c) de su interior (i): $r(S) = c(i(S))$
- Un **conjunto** es **regular** si es igual a su regularización: $S = r(S)$



Operaciones de sólido

Las operaciones booleanas de sólidos se definen en base a la clausura (c) del interior (i):

- **Unión:** $A \oplus B = c(i(A \cup B))$
- **Diferencia:** $A \ominus B = c(i(A - B))$
- **Intersección:** $A \odot B = c(i(A \cap B))$

\cup , $-$ y \cap son operadores en el conjunto.

El resultado de la resta de los cubos anteriores sería, por tanto, vacía.

Propiedades del sólido

Propiedades del sólido:

- Cerrado.
- Orientable.

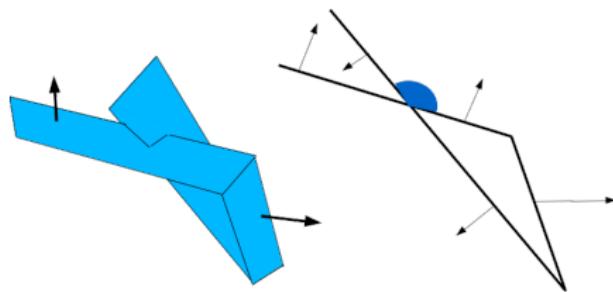


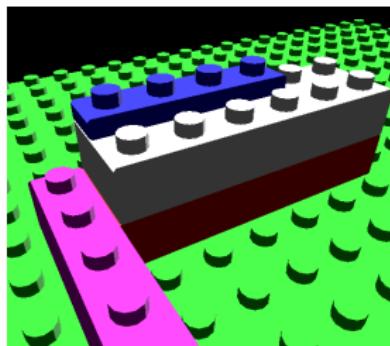
Figura 4: Podemos saber la orientación de forma automática, mirando hacia el interior del sólido.

Construcción de un sólido

Un sólido puede construirse de varias formas (difíciles de construir):

- Instanciación de primitivas (ej. cubos).
- Descomposición espacial (ej. celdas, octrees, ...).
- Geometría constructiva de sólidos (CSG).
- Barrido (ej. solevados, ...).
- Fronteras (Brep).

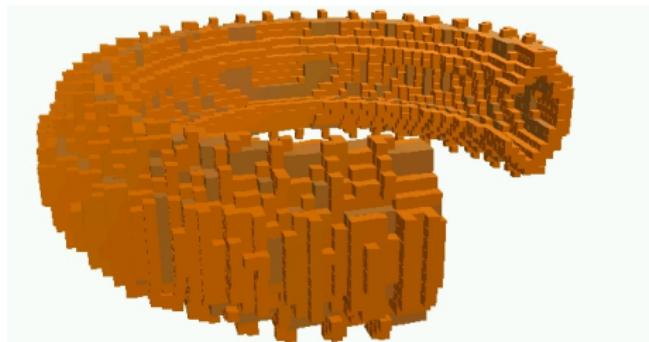
Instanciación de primitivas



El objeto se representa como un conjunto de primitivas que se instancian en el escenario:

- Las primitivas se pueden intersectar.
- No se pueden calcular propiedades.

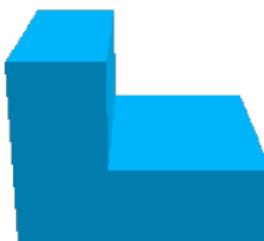
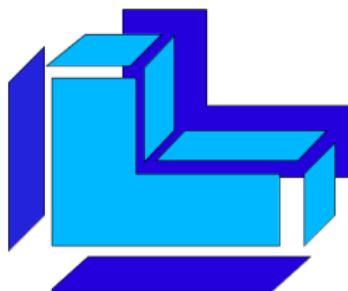
Descomposición espacial



El sólido se describe en base a una descomposición del espacio en una colección de celdas (jerárquica o no):

- Las celdas son elementos simples disjuntos.
- El forma mas común de celda es la cúbica.
- La representación de un sólido es la enumeración de las celdas que ocupa.

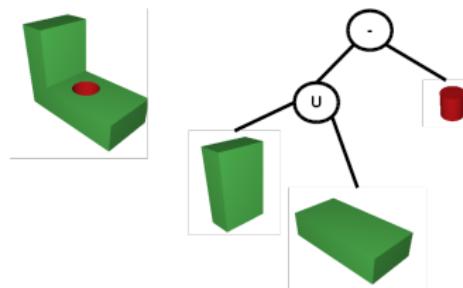
Representación de fronteras B-rep



El objeto se representa mediante un conjunto de caras que describen su frontera:

- Cuando se representa un sólido la frontera debe ser cerrada y orientable.
- La mayor parte de los sistemas utilizan representaciones de fronteras, con caras poligonales.

Geometría Constructiva de Sólidos (CSG)



El sólido se representa como una expresión booleana que indica como construirlo a partir de primitivas simples:

- La representación es la propia expresión.
- Las primitivas son sólidos parámetricos.
- La representación no es única.

Muy usada en motores de realidad virtual y juegos por su versatilidad.

Mallas de polígonos

Las mallas de polígonos se basan en tres primitivas: vértice (V), arista (E) y cara (F).

La **malla** es una terna (V, E, F) donde:

- $V = \{p_i \in \mathbb{E}^3 \mid 1 \leq i \leq n\}$
- $E = \{(i, j) \in V \times V\}$
- $F = \{(i_1, i_2, \dots, i_n) \mid i_k \in V, (i_k, i_{k+1}) \in E\}$

Donde:

- Cada arista E pertenece al menos a una cara F .
- Cada vértice V pertenece al menos a una arista E .

Mallas de polígonos

- Una arista es frontera si pertenece a una sola cara.
- Una malla es manifold (variedad) si:
 - ▶ Cada arista pertenece a lo sumo a dos caras
 - ▶ Para cada vértice sus polígonos adyacentes forman un disco

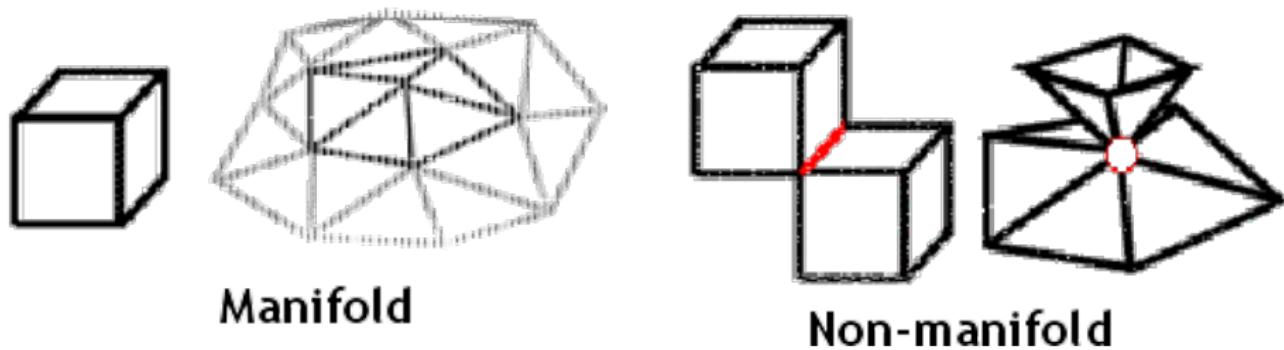


Figura 5: Diferencias entre mallas manifold y no-manifold.

Poliedros

Una malla es un **poliedro** si:

- Es manifold.
- Es cerrado.
- No hay intersecciones entre caras (salvo en aristas y vértices compartidos).

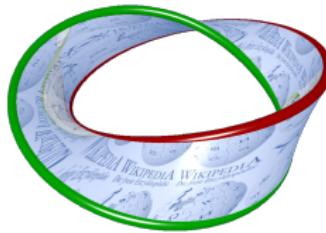
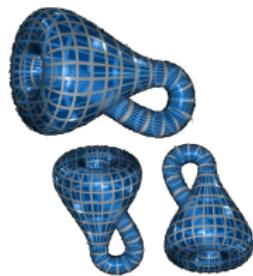
A cada cara se le puede asignar una **orientación**:

- La orientación se determina por el recorrido de sus vértices.
- Puede ser horario (CW) o antihorario (CCW).
- El lado exterior (o frontal) de la cara se determina por convenio (suele ser CCW).
 - ▶ Aunque la orientación de una cara se puede forzar.

Relación entre mallas y sólidos

Una malla es **orientable** si todas sus caras se pueden orientar de forma consistente.

- Todas con orientación CW o CCW.
- Cada arista se recorre en sentido contrario en sus dos caras.



Una **malla cerrada orientable** representa un sólido.

Mallas: estructuras de programación (I)

```
var vertex_array = [] # Vértices
```

```
var index_array = [] # Caras
```

```
vertex_array[0] = Vector3(-1, -1, 0)
```

```
vertex_array[1] = Vector3(-1, 1, 0)
```

```
vertex_array[2] = Vector3(1, 1, 0)
```

```
vertex_array[3] = Vector3(1, -1, 0)
```

```
index_array[0] = 0
```

```
index_array[1] = 1
```

```
index_array[2] = 2
```

```
index_array[3] = 3
```

```
index_array[4] = 0
```

Mallas: estructuras de programación (II)

```
vertex_array[0] = Vector3(-1, -1, 0)
vertex_array[1] = Vector3(-1, 1, 0)
vertex_array[2] = Vector3(1, 1, 0)
vertex_array[3] = Vector3(1, -1, 0)
```

Si no le damos un índice a las caras se asumen que son triángulos (CCW).

En Godot hay cuatro formas de hacer geometría mediante código:

https://docs.godotengine.org/en/3.2/tutorials/content/procedural_geometry/index.html?highlight=geometry

Ejercicio Teórico-Práctico:

Creación de un cubo proceduralmente.

Crea los arrays necesarios para dibujar las caras de un cubo. Revisa el tutorial de Godot para crear geometría, ignora las normales (*normals*) y las coordenadas de textura (*UV*). Introduce el código en un nodo de Godot del tipo que consideres más apropiado.

Tema 2 - Arquitectura y modelos para entornos virtuales.

2.3 Sistemas básicos de iluminación y cámaras.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Contenido del tema

Tema 2: Arquitectura y modelos para entornos virtuales.

- 2.1 Grafos de escena y modelos jerárquicos.
- 2.2 Métodos básicos de representación.
- 2.3 Sistemas básicos de iluminación y cámaras.
- 2.4 Modelos de generación procesal.

2.1 Sistemas básicos de iluminación y cámaras

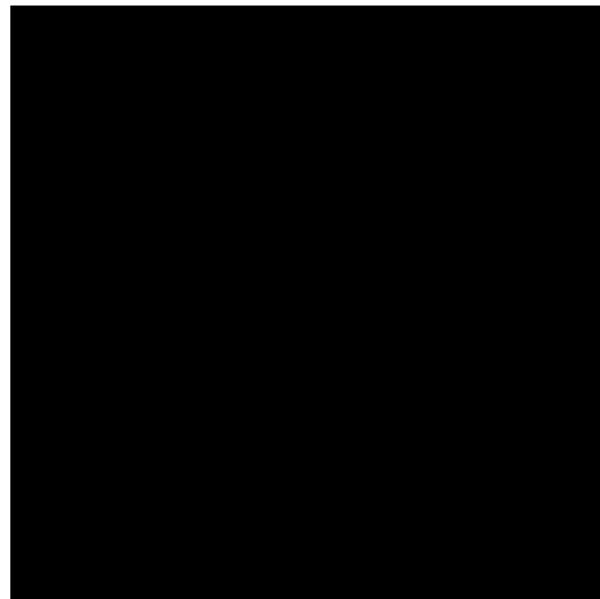
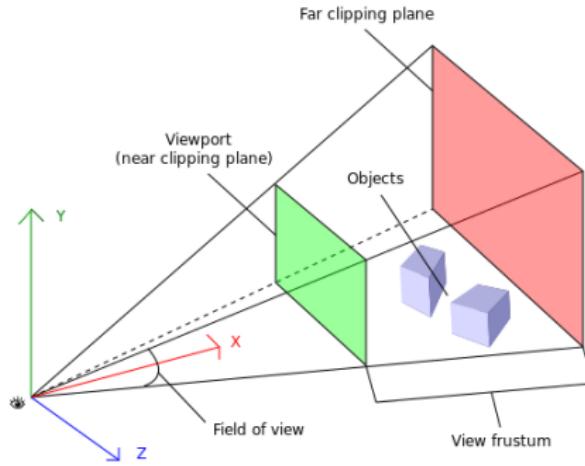


Figura 1: Una escena sin iluminación o cámara.

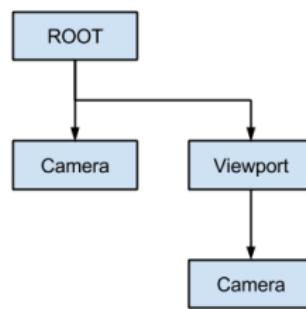
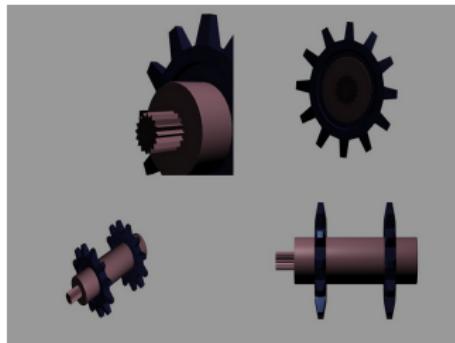
Cámaras (I)

- Cámara y pantalla no son lo mismo.
- **Vista:** (viewport) la vista es el área visible de la cámara activa. Suele estar ligado a un *buffer*, por lo que tiene una resolución fija en píxeles.



Cámaras (II)

- Una cámara está asociada a una vista (normalmente su vista padre).
- Solamente puede haber una cámara activa.
- Las cámaras pueden tener sub-vistas asociadas (sub-viewport), que suelen renderizar a texturas.



Ejercicio Teórico-Práctico:

Creación de un cámara FPS.

Estudia el siguiente tutorial:

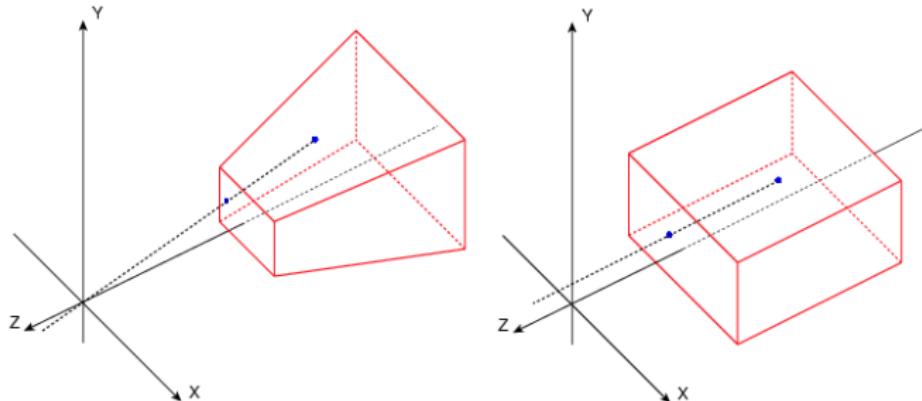
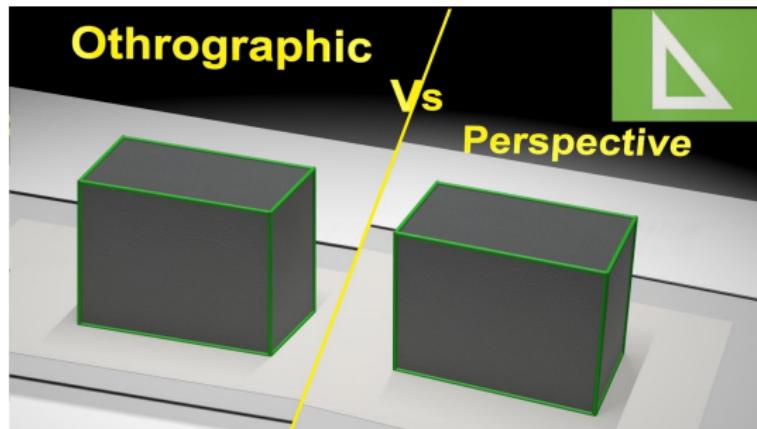
[https:](https://docs.godotengine.org/en/stable/tutorials/3d/fps_tutorial/part_one.html)

//docs.godotengine.org/en/stable/tutorials/3d/fps_tutorial/part_one.html

Plantea el movimiento de cámara con raton en Godot. Implementa un botón de salto sin utilizar física de Godot, para ello estudia el movimiento de tiro parabólico.

https://en.wikipedia.org/wiki/Projectile_motion

Perspectiva vs. ortogonal



La física de la luz (I)

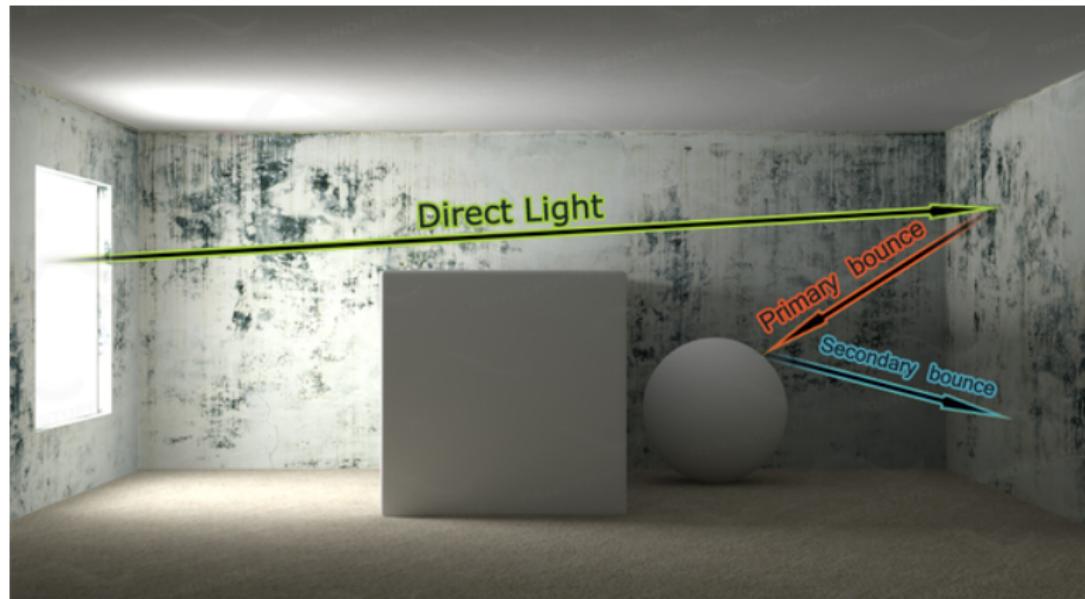


Figura 3: Ejemplo de rebotes de luz directos e indirectos.

La física de la luz (II)



Figura 4: Ejemplo de difracción de la luz.

Modelo sencillo de iluminación: fuentes de luz

Las fuentes de luz son la forma más sencilla de iluminación:

- Fuente puntual: $\vec{p} = (p_x, p_y, p_z)$. Emiten luz en todas direcciones.
- Fuente direccional: $\hat{d} = (d_x, d_y, d_z)$. Emite luz en una única dirección (normalizada). Está situada en el infinito.
- Fuente focal: Viene dada por posición y dirección, y tiene un ángulo de apertura. $F \in \mathbb{E}^3 \times \mathbb{R}^3 \times \mathbb{R}$

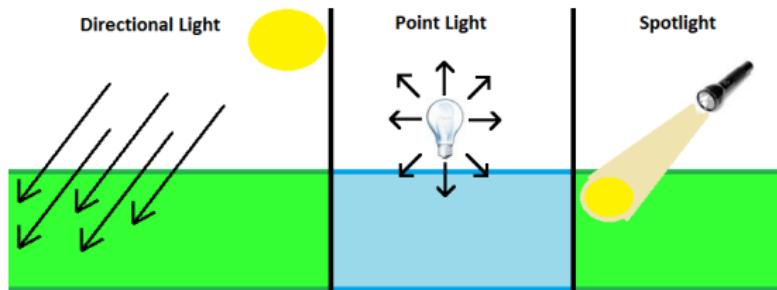


Figura 5: Tipos de luces básicas.

Funcionamiento de la iluminación básica

Una *normal de una cara* es el vector unitario en \mathbb{R}^3 que sale perpendicular a esta.

- La orientación de la cara normalmente indica el sentido del vector de dirección.
- Un vértice tiene varias normales, correspondiente a cada una de sus caras.
 - ▶ En los entornos de RV las normales se suelen definir a nivel de vértices (**normal por vértice**).

normal_array[0] = Vector3(0, 0, 1)

Lambert: el modelo de iluminación más sencillo

Para cada punto de la superficie:

$$c = \hat{L} \cdot \hat{N} = |\vec{N}| |\vec{L}| \cos \alpha = \cos \alpha$$

Nota como los vectores de dirección de luz y normales están **normalizados**.

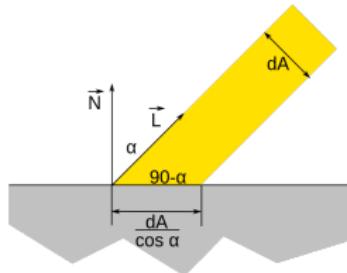


Figura 6: Modelo de lambert: luz difusa.

Materiales sencillos e iluminación

- Pero la superficie puede tener un material. El material más sencillo es un color.
- Un **color** c se define como un vector en \mathbb{R}^3 , con tres componentes: rojo, verde y azul. Aunque el dominio visible está en $[0, 1]$, a veces se permite exceder este valor (ej. materiales de emisión).
- Los colores se mezclan lumínicamente, dos colores se pueden mezclar entre sí multiplicando componente a componente ambos colores.

Entonces:

$$c = c_{material} \cdot \hat{L} \cdot \hat{N} = c_{material} \cdot |\vec{N}| |\vec{L}| \cos \alpha = c_{material} \cdot \cos \alpha$$

- Si hay varias luces, entonces se suman los valores.

Propiedades de las luces

Las luces suelen extenderse añadiendo propiedades numerosas:

- **Color.**
- **Energía:** un multiplicador a la intensidad (el color ya no es normalizado en ese caso).
- **Energía indirecta:** segundo multiplicador usado para la luz indirecta y ambiental.
- **Luz negativa:** la luz se vuelve sustractiva en lugar de aditiva.
- **Especular:** afecta a la intensidad del brillo que se produce en determinados materiales.
- **Cocinado (bake):** las luces se pueden precalcular para aumentar el rendimiento.
- etc.

Sombras arrojadas

En los mundos virtuales las luces son las que arrojan sombras (opcional).

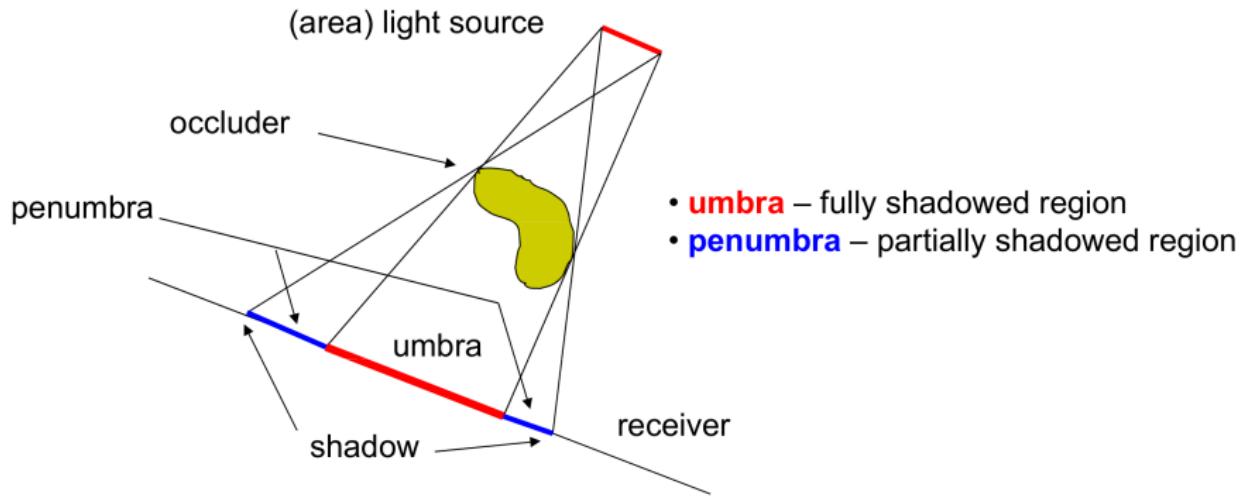


Figura 7: Esquema de sombra arrojada por una luz.

Sombras arrojadas terminología

Definiciones:

- **Receptor:** (*receiver*) objeto que está bajo la sombra.
- **Oclusor:** (*caster/occluder*) objeto que bloquea la luz del receptor.
- **Umbra:** región que está completamente bajo una sombra.
- **Penumbra:** región que está en transición entre la umbra y un área iluminada.

Tipos de sombra según la luz (I)

Fuentes puntuales, direccionales y algunas focales:

- Las sombras son duras si no hay iluminación global de escena (no hay penumbra).
- Son rápidas de calcular.

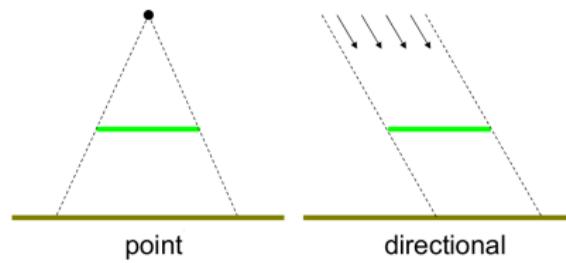


Figura 8: Sombras generadas por luces puntuales y direccionales.

Tipos de sombra según la luz (II)

Sombras en luces de área (se usa un rectángulo y múltiples vectores en bordes) y algunas focales (se pueden usar múltiples vectores):

- Las sombras son suaves incluso si no hay iluminación global de escena.
- Son algo más lentas de calcular.

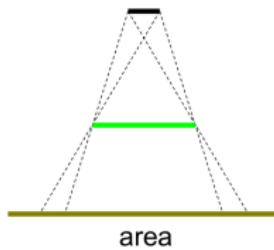


Figura 9: Sombras generadas por luces de área.

Técnicas para el cálculo de sombras

Técnicas más comunes para el cálculo de sombras en tiempo real:

- *Planar Shadows.*
- *Shadow Volume.*
- *Shadow Maps.*
- *Ray casting.*

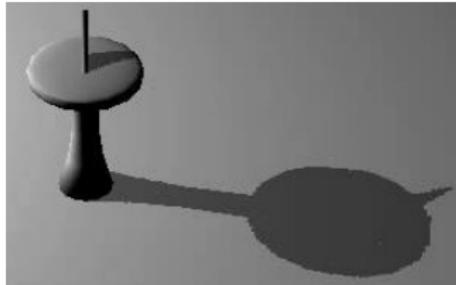
Planar Shadows (I)

- Se utiliza una matriz de proyección.
- Dado un punto \vec{p} del plano, y su normal \hat{n} , calculamos la ecuación del plano: $d = -(\hat{n} \cdot \vec{p})$, $\hat{n} \cdot \vec{p}_i + d = 0$.
- A partir del plano, y un punto de proyección p , creamos una matriz que proyecte un punto arbitrario en ese plano:

$$M = \begin{pmatrix} \hat{n} \cdot \vec{p} + d - p_x n_x & -p_x n_y & -p_x n_z & -p_x d \\ -p_y n_x & \hat{n} \cdot \vec{p} + d - p_y n_y & -p_y n_z & -p_y d \\ -p_z n_x & -p_z n_y & \hat{n} \cdot \vec{p} + d - p_z n_z & -p_z d \\ -n_x & -n_y & -n_z & \hat{n} \cdot \vec{p} \end{pmatrix}$$

Planar Shadows (II)

- Esta matriz M es una transformación homogénea, y se inserta después de calcular el espacio objeto-mundo, y antes de la transformación mundo-ojo.
- Algoritmo:
 - 1 Deshabilitar el *z-buffer* y dibujar la sombra en el *alpha-buffer*.
 - 2 Habilitar el *z-buffer* y dibujar el objeto.
 - 3 Dibujar el suelo y modular el canal *alpha* de destino.
- Problema: sombras sin penumbra.



Shadow volume (I)

- Un volumen de sombra es un espacio formado por un objeto oclusor y la luz.
- Limitado por las aristas del objeto oclusor.
- Todos los objetos dentro del volumen están en sombra.

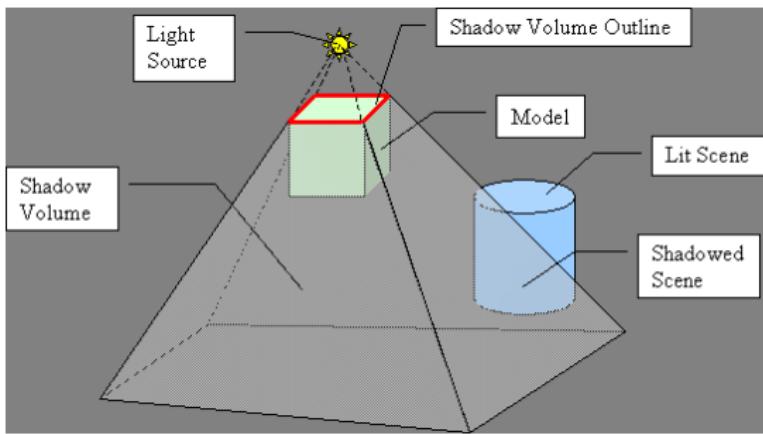


Figura 10: Esquema que define el volumen de sombra.

Shadow volume (II)

Algoritmo:

- ① Crear un contador.
- ② Lanzar un rayo en la escena.
- ③ Incrementar el contador cuando el rayo perfore una cara frontal.
- ④ Decrementar el contador cuando el rayo perfore una de las caras traseras del volumen de sombra.
- ⑤ Cuando golpee un objeto:
 - ▶ Si contador > 0 → en sombra.
 - ▶ En otro caso → no está en sombra.

Shadow volume (III)

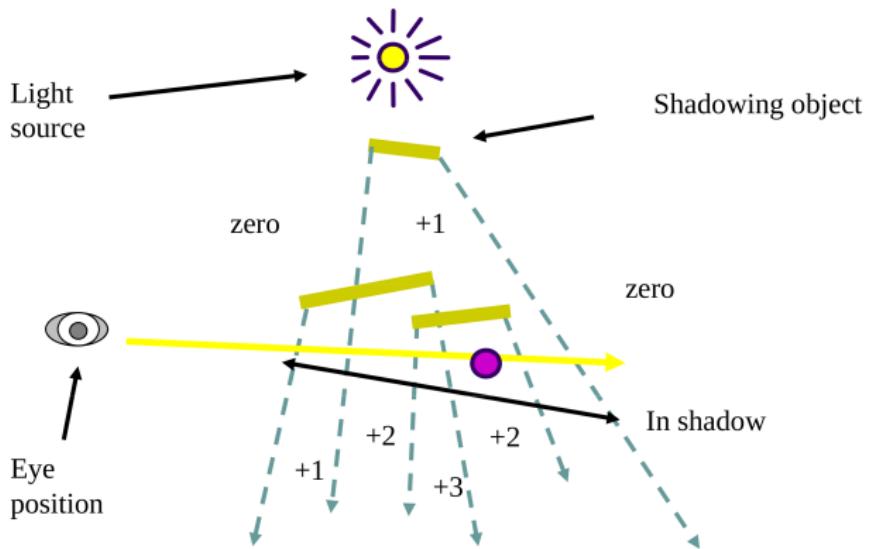


Figura 11: Ejemplo de algoritmo para Shadow volumes.

Shadow volume (IV)

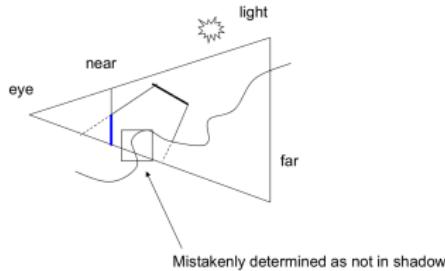
Implementación: solución *z-pass*

- ① Renderiza la escena solamente con luz ambiente y emisión, se actualiza el *z-buffer*.
- ② Desactivamos el *z-buffer* y la escritura en bufferes de color. Activamos el *stencil buffer* (manteniendo el test de profundidad activo).
- ③ Inicializamos el *stencil buffer* a 0.
- ④ Dibujamos el volumen de sombras dos veces (usando *face culling*).
 - ▶ 1^a pasada: renderizamos las caras frontales, +1 *stencil buffer* si pasa el test de profundidad.
 - ▶ 2^a pasada: renderizamos las caras traseras, -1 *stencil_buffer* si pasa el test de profundidad.
- ⑤ Renderizamos la escena de nuevo, con la luz de la fuente cuando los píxeles valen 0 en el stencil. (Si en un pixel $\text{stencil} \leq 0 \rightarrow$ en sombra, en otro caso iluminado.)

Shadow volume (V)

Problemas del z-pass:

- ① Cuando el plano cercano intersecta con alguna cara del volumen de sombras (corta la cara).
- ② Cuando el ojo está en el volumen de sombras
 - ▶ contador = 0 ya no implica sombra → deberíamos tener en cuenta el nº de volúmenes en los que estamos.



Shadow volume (VI)

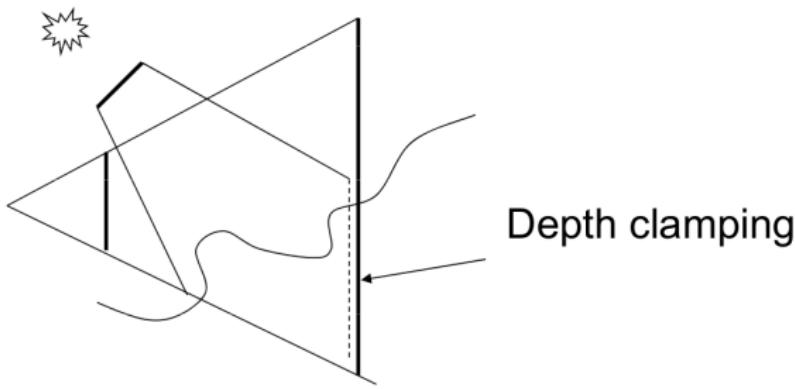
Implementación: solución *z-fail*

- ① Renderiza la escena solamente con luz ambiente y emisión, se actualiza el *z-buffer*.
- ② Desactivamos el *z-buffer* y la escritura en bufferes de color. Activamos el *stencil buffer* (manteniendo el test de profundidad activo).
- ③ Inicializamos el *stencil buffer* **según el punto de vista**.
- ④ Dibujamos el volumen de sombras dos veces (usando *face culling*).
 - ▶ 1^a pasada: renderizamos las caras **traseras**, +1 *stencil buffer* si **falla** el test de profundidad.
 - ▶ 2^a pasada: renderizamos las caras **delanteras**, -1 *stencil_buffer* si **falla** el test de profundidad.
- ⑤ Renderizamos la escena de nuevo, con la luz de la fuente cuando los píxeles valen 0 en el stencil. (Si en un pixel $\text{stencil} \leq 0 \rightarrow$ en sombra, en otro caso iluminado.)

Shadow volume (VII)

Problemas del *z-fail*:

- 1 La sombra puede penetrar el plano trasero → se puede poner un máximo a la distancia en el *z-buffer*, cercano al volumen de sombras.



Shadow maps (I)

- Determinar si el objeto es visible:
 - ▶ Usar el algoritmo de z-buffer anterior, pero la cámara hace de luz.

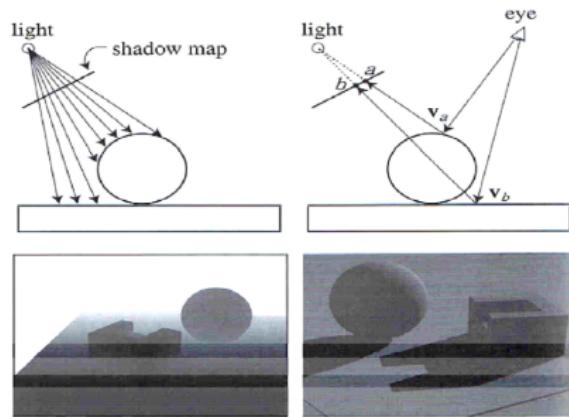
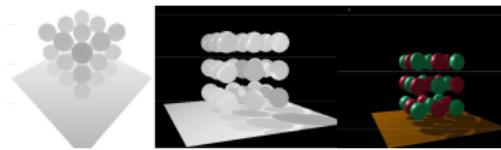


Figura 12: Algoritmo de mapa de sombras.

Shadow maps (II)

Algoritmo:

- ① Renderizar la escena usando la luz como cámara y almacenando el *z-buffer*.
- ② Generar un buffer de luz (textura 2D) a partir del *z-buffer* (llamado *shadow map*).
- ③ Renderizar la escena usando la cámara y con el *z-buffer* activo
 - ▶ Para cada fragmento visible en espacio local, transformarlo al espacio de la luz (matrices *modelview*, *projection* y *shadowmatrix*):
 $(x, y, z) \rightarrow (x', y', z')$.
 - ▶ Comparar z' con $z = shadow_map(x', y')$, si $z' \leq z$ (es más cercano que la luz), entonces el pixel no está en la sombra, en caso contrario sí.



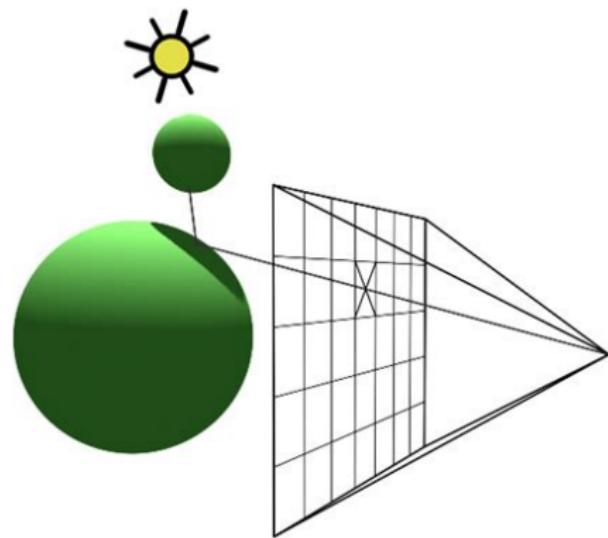
Shadow maps (III)

Problemas:

- Depende de la resolución de la textura (*aliasing*).
- Resolución del *z-buffer*, normalmente 8-24 bits de precisión (depende de la GPU).
- Aliasing en sombras del objeto oclusor producidas por el mismo objeto oclusor (*self-shadow*).
- Las sombras aparecen como por bloques (pixelado).
 - ▶ Especialmente problemático cuando un pixel del mapa de sombras cubre varios píxeles de la pantalla.

Ray casting (I)

- Se lanzan rayos: $\vec{r}(t) = \vec{o} + t\vec{d}$, con $0 \leq t \leq \infty$
- Se simulan rayos de luz, que viajan siempre en línea recta, hasta que rebotan (se usa la normal de la superficie).

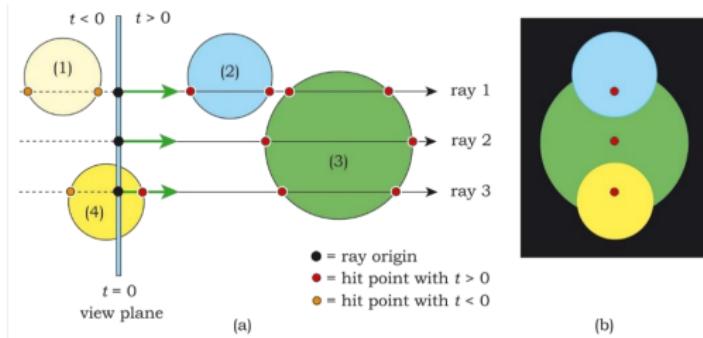


Ray casting (II)

Algoritmo:

- ① Los rayos se lanzan desde cada píxel de la cámara.
- ② Si un rayo choca contra un objeto, se lanzan rayos hacia las fuentes de luz.
 - ▶ Si alguna de estos rayos choca contra un objeto, está bajo sombra.

Permite hacer sombras translúcidas, incluso de colores.



Iluminación global (I)



Figura 13: Ejemplo de iluminación global.

Iluminación global (II)

Algoritmos en tiempo real (o casi):

- *Ray Tracing* y derivados.
 - ▶ Koskela, M., Immonen, K., Mäkitalo, M., Foi, A., Viitanen, T., Jääskeläinen, P., ... & Takala, J. (2019). **Blockwise multi-order feature regression for real-time path-tracing reconstruction.** ACM Transactions on Graphics (TOG), 38(5), 1-14.
https://www.youtube.com/watch?v=julyJEUVdBI&feature=emb_logo
 - ▶ Majercik Z., Guertin J.P., Nowrouzezahrai D., MC Guire M.: **Dynamic diffuse global illumination with ray-traced irradiance fields.** Journal of Computer Graphics Techniques Vol 8, 2 (2019)

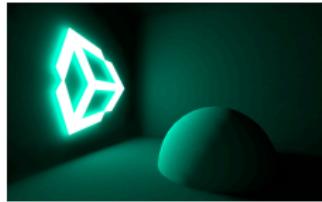
Iluminación global (III)

Algoritmos en tiempo real (o casi):

- *Radiosity* y derivados.
 - ▶ MC Taggart G.: **Half-life 2/valve source shading.** In Game Developer's Conference (2004), vol. 1. 1
- *Signed Distance Fields.*
 - ▶ Hu, J., Yip, M., Alonso, G. E., Gu, S., Tang, X., & Jin, X. (2020). **Signed Distance Fields Dynamic Diffuse Global Illumination.** arXiv preprint arXiv:2007.14394. <https://godotengine.org/article/godot-40-gets-sdf-based-real-time-global-illumination>
https://www.youtube.com/watch?v=ztkBRFocHww&feature=emb_logo

Iluminación y visualización

- Materiales, algoritmos de visualización e iluminación están íntimamente ligados.



Tema 2 - Arquitectura y modelos para entornos virtuales.

2.4 Modelos de generación procesal.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Contenido del tema

Tema 2: Arquitectura y modelos para entornos virtuales.

- 2.1 Grafos de escena y modelos jerárquicos.
- 2.2 Métodos básicos de representación.
- 2.3 Sistemas básicos de iluminación y cámaras.
- 2.4 Modelos de generación procesal.

2.4 Modelos de generación procesal

Hay varios objetos que se pueden generar mediante algoritmos que basados en sólidos o mallas.

- Revolución (*lathe*).
- Extrusión y soleados (*extrusion/loft*).
- Operaciones booleanas (*boolean*).

Revolución (I)

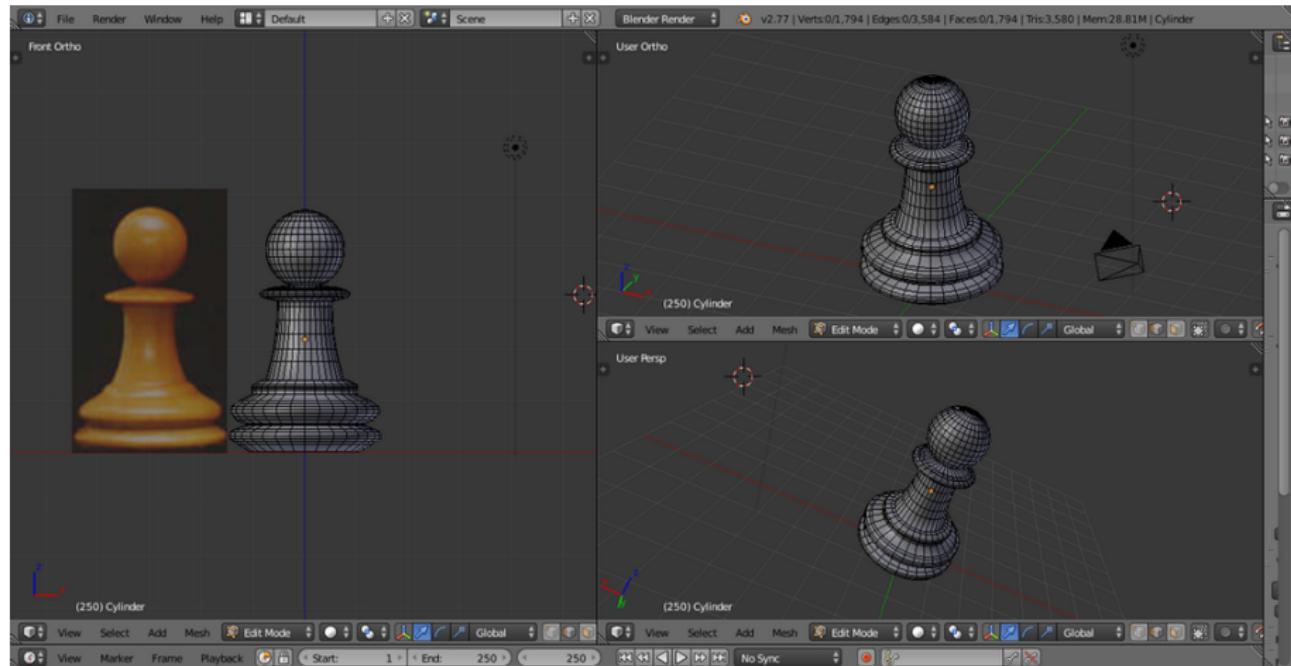


Figura 1: Objeto creado por revolución.

Revolución (II)

- 1 Tomar el vector del eje.
- 2 Desplazar un vértice del centro del eje y rotar a incrementos (n^o de caras) hasta completar el n^o de grados deseado (normalmente 360^o).
- 3 Siguiente vértice:
 - ▶ Añadir cierta altura ($1/altura_total_modelo$)
 - ▶ Si la iteración es par, sumar ángulo de giro, en otro caso restar. Repetir 2.
- 4 Construir las caras: $(v_{j,i}, v_{(j+1) \bmod N, i}, v_{j, (i+1) \bmod N})$, N es el numero de pasos.

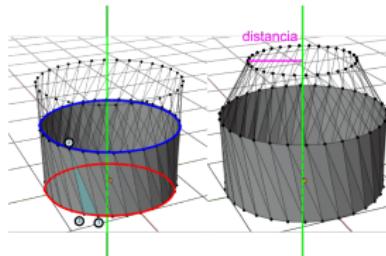


Figura 2: Pasos para realizar la revolución.

Revolución (III)

Matriz de rotación en eje arbitrario:

<http://ksuweb.kennesaw.edu/~plaval/math4490/rotgen.pdf>

$$T = \begin{pmatrix} tu_x^2 + C & tu_x u_y - Su_z & tu_x u_z + Su_y & 0 \\ tu_x u_y + Su_z & tu_y^2 + C & tu_y u_z - Su_x & 0 \\ tu_x u_z - Su_y & tu_y u_z + Su_x & tu_z^2 + C & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde:

- $\hat{r} = (u_x, u_y, u_z)$ = eje de rotación, y Θ ángulo de rotación (360°).
- $C = \cos \Theta$
- $S = \sin \Theta$
- $t = 1 - C$

Extrusión y solevados (I)

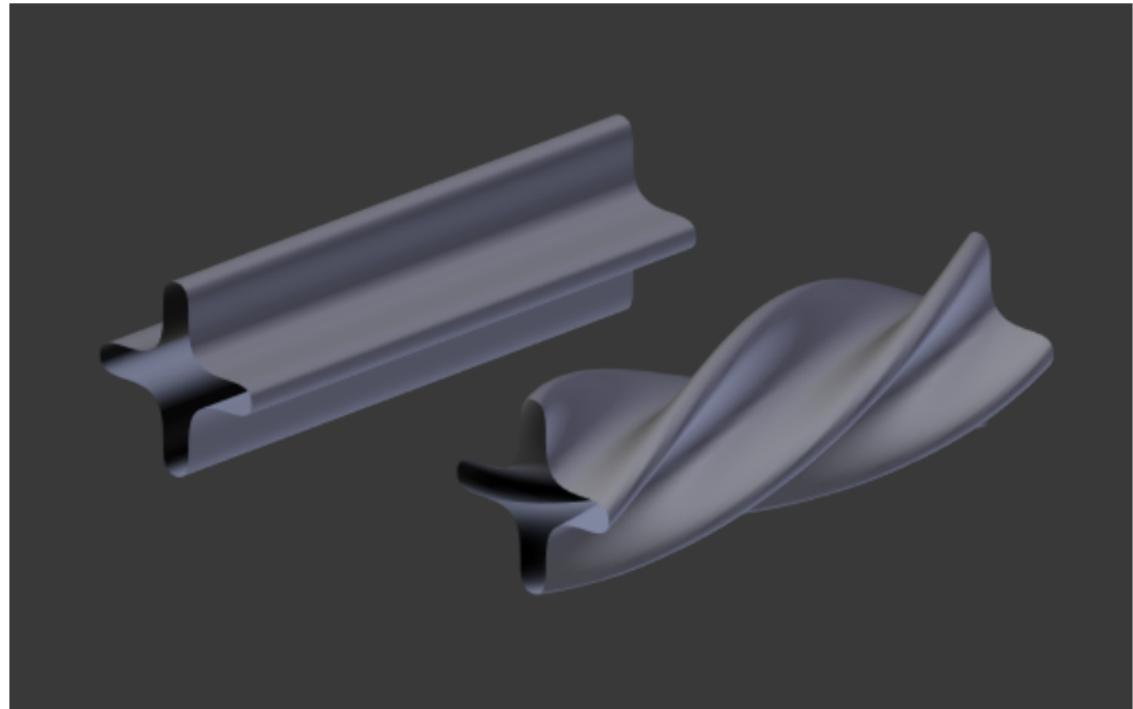
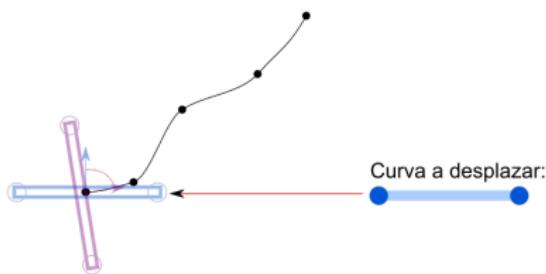


Figura 3: Objeto creado mediante extrusión (izquierda) y solevado (derecha).

Extrusión y soleados (II)

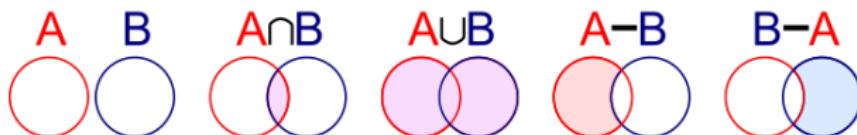
- 1 $i = 0$
- 2 \vec{k}_i es el primer vértice de la curva de recorrido.
- 3 $\hat{d} = \text{norm}(k_{i+1} - k_i)$.
- 4 Desplazar la curva al punto \vec{k} , girarla Θ grados (punto de pivote):
$$\Theta = \arccos[(\hat{d} \cdot \hat{N}) / (|\hat{d}| \cdot |\hat{N}|)] = \arccos(\hat{d} \cdot \hat{N})$$
- 5 Incrementar i y repetir 2.
- 6 Construir los triángulos como en la revolución.



Operaciones booleanas (I)

Conjunto de operaciones booleanas entre sólidos en \mathbb{R}^3 :

- Unión: $A \cup B = \{x \in A \text{ or } x \in B\}$
- Intersección: $A \cap B = \{x \in A \text{ and } x \in B\}$
- Diferencia: $A - B = \{x \in A \text{ and } x \notin B\}$



Operaciones booleanas (II)

Algoritmo:

- 1 Encontrar las intersecciones entre A y B .
- 2 Dividir ambos modelos por las intersecciones, y clasificar las regiones.
- 3 Decidir qué regiones se borran.
- 4 Unir entre sí las regiones restantes.

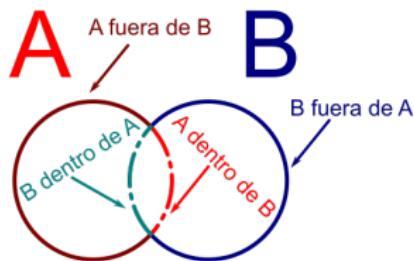


Figura 4: Clasificación de regiones.

<https://www.youtube.com/watch?v=QWtknlm5kn8>

Operaciones booleanas (III)

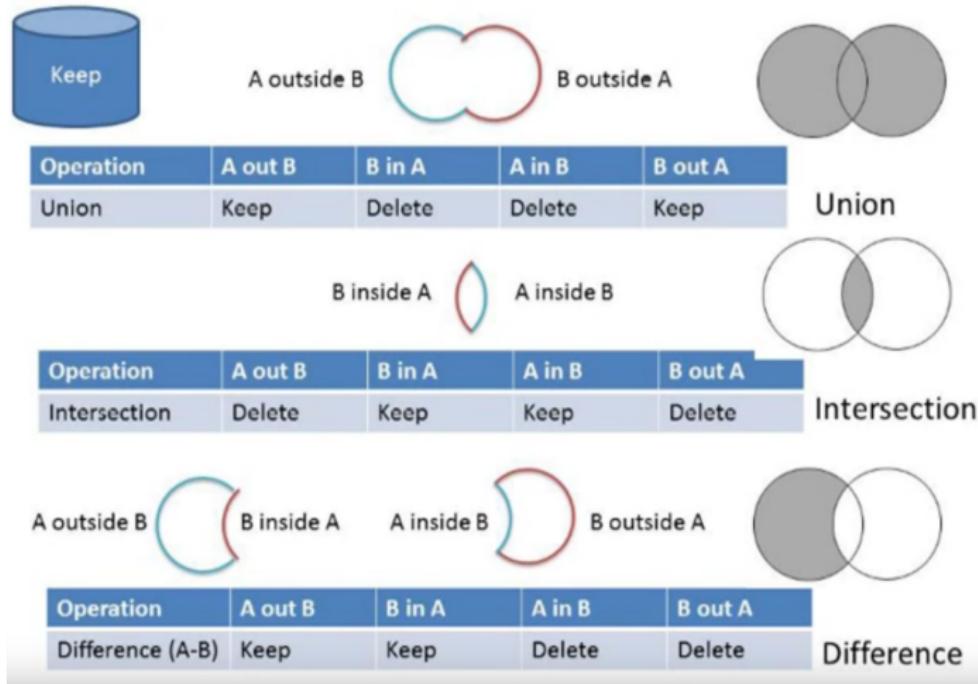


Figura 5: Esquema de decisiones para eliminar piezas.

Modelado procedural

- **Modelado procedural:** el modelo 3D se describe usando código en lugar de datos

Ventajas:

- Generación automática.
- Fácil parametrización y continuidad.
- Menor tamaño en memoria y disco.
- Mayor cantidad de contenido.
- Variedad (aleatoriedad).

Generación de números pseudoaleatorios

Se basan en una **semilla**, siempre obtendremos los mismos valores **si el algoritmo y la semilla coinciden**.

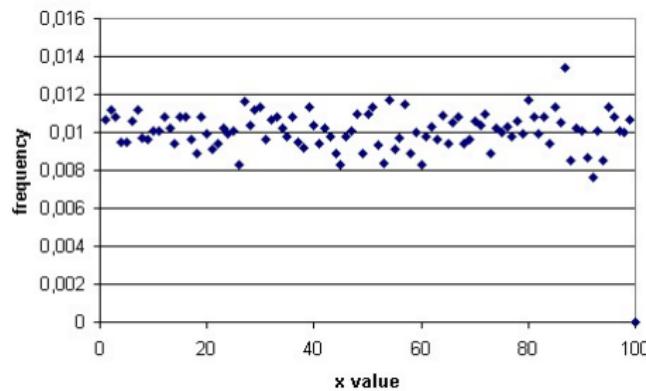


Figura 6: Generación de números pseudoaleatorios.

Ejemplo: Perlin Noise

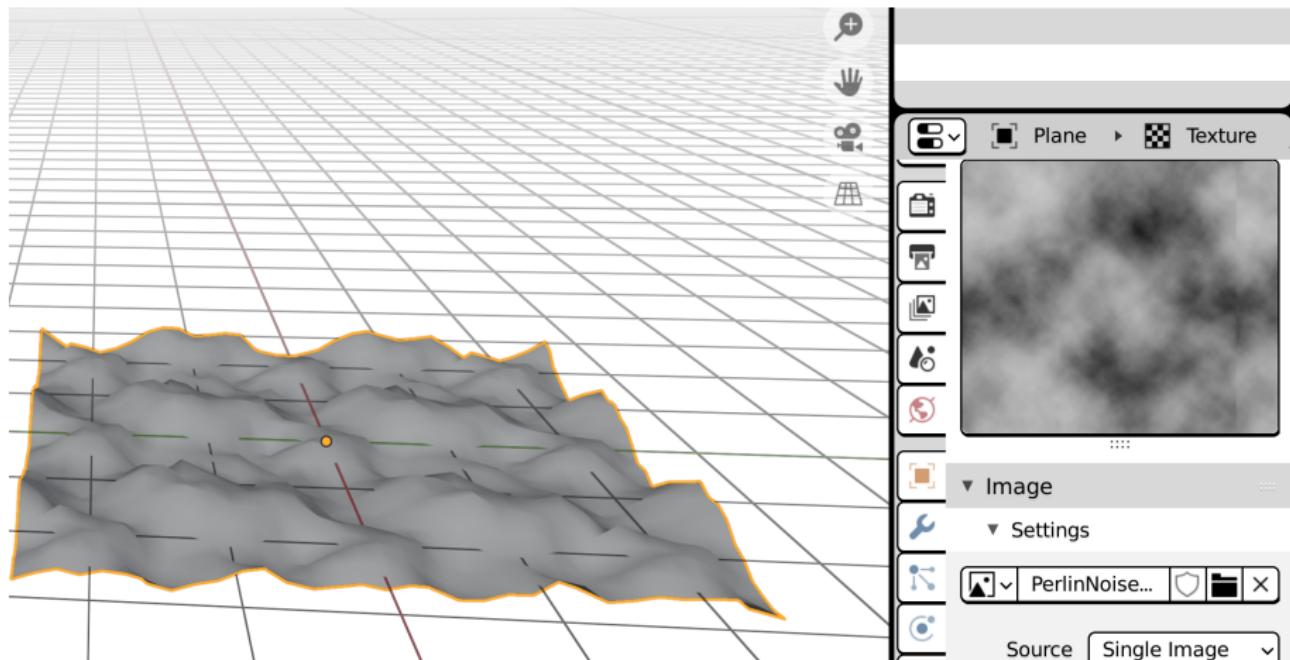
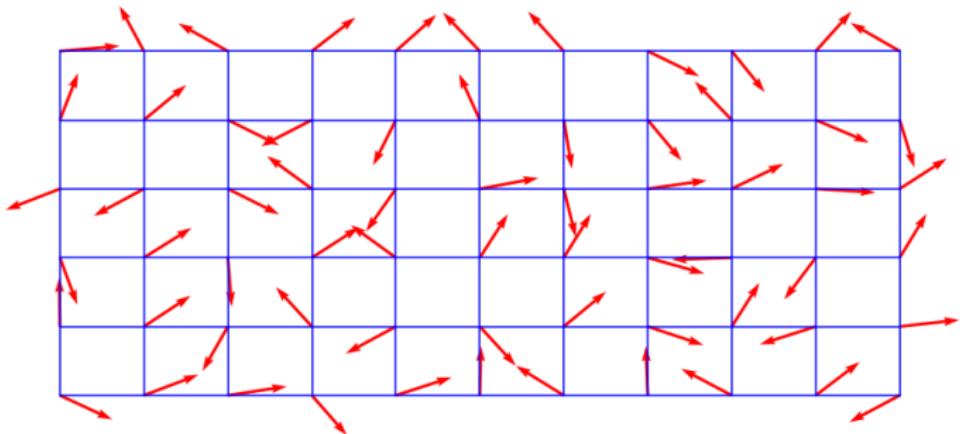


Figura 7: Ejemplo de ruido Perlin 2D.

Algoritmo de Perlin (I)

- ➊ Definir una rejilla de $N > 1$ dimensiones:
 - Cada intersección está asociada con un vector de valores aleatorios.



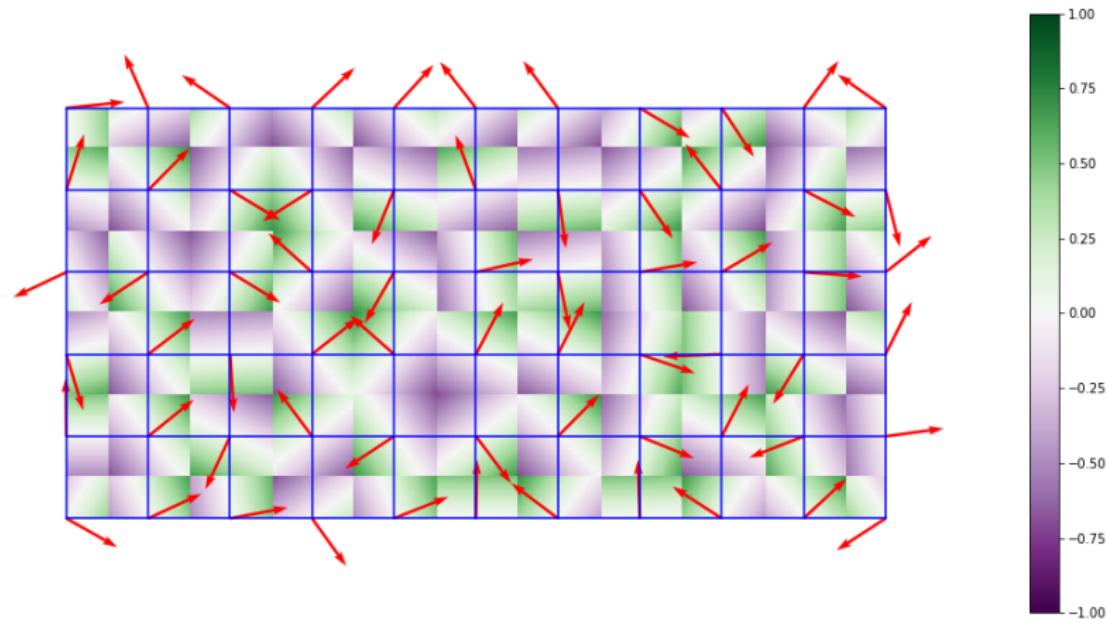
Algoritmo de Perlin (II)

② Para calcular el valor de cada punto:

- Cada punto de intersección, desplazarse en la dirección aleatoria dada y ver en que celda cae.
- Identificar los puntos de las esquinas de la celda y sus direcciones.
- Calcular el vector de desplazamiento (desde el punto candidato a cada esquina).

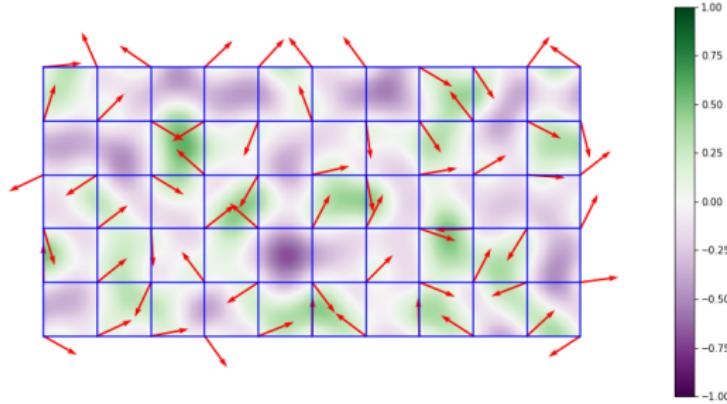
Algoritmo de Perlin (III)

- Calcular el producto escalar del vector de gradiente y los vectores de desplazamiento.



Algoritmo de Perlin (IV)

- ③ El último paso es la interpolación de los valores de las esquinas.
- En muchos casos la función es de tipo sigmoide (*smoothstep*), pero cualquier función cuya primera derivada (y posiblemente la segunda) sean 0 es válida. Por ello, se suele aproximar a una interpolación linear según avanzamos hacia las esquinas.



Fractales determinísticos

- Se parte de una forma inicial (iniciador) y una regla de sustitución.
- En cada interacción se sustituye una parte usando la regla de sustitución.



Figura 8: Ejemplo de fractal determinístico.

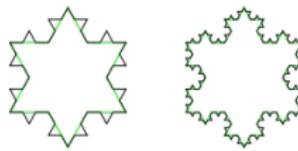
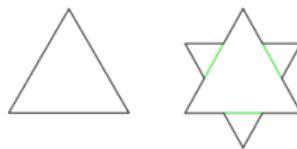


Figura 9: Otro ejemplo de fractal determinístico.

Generación de terrenos sencillos automáticos

La regla puede contener números aleatorios (ej. Perlin Noise):

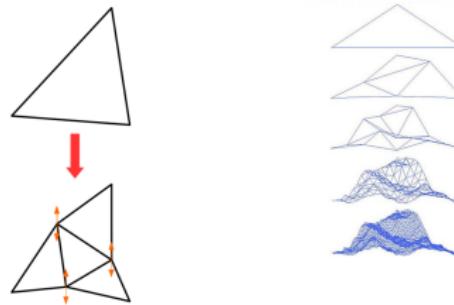


Figura 10: Generación de terreno mediante fractal determinístico.

https://upload.wikimedia.org/wikipedia/commons/6/6d/Animated_fractal_mountain.gif

Gramáticas formales

Una gramática formal consiste en una serie de **reglas de producción** ($A \rightarrow B$), donde cada lado contiene una serie de símbolos.

- **Símbolo inicial:** si podemos reducir hasta aquí significa que los símbolos pertenecen al lenguaje.
- **Símbolos no terminales:** puede aplicarse alguna regla de producción.
- **Símbolos terminales:** no se puede aplicar más reglas.
- **Cadena vacía:** (ϵ) representa la ausencia de símbolos, y se considera terminal.

Ejemplo:

$$S \rightarrow AB$$

$$S \rightarrow \epsilon \text{ (de forma simplificada: } S \rightarrow AB|\epsilon)$$

$$A \rightarrow aS$$

$$\$B \rightarrow b$$

Clasificación de gramáticas (I)

La clasificación de Chomsky define 4 tipos de gramáticas.

Tipos de grámáticas:

- Tipo-0: incluyen todas las gramáticas formales.
- Tipo-1: generan lenguajes sensibles al contexto.
- Tipo-2: generan lenguajes libres de contexto (contexto teórico para la mayoría de los lenguajes de programación).
- Tipo-3: generan lenguajes regulares (expresiones regulares de búsquedas [grep]).

Clasificación de gramáticas (II)

Significado de los símbolos de las tablas:

- a = Terminal; A, B = No terminales.
- α, β, γ = cadenas de terminales y/o no terminales.
 - ▶ α, β pueden ser ϵ
 - ▶ $\gamma \neq \epsilon$

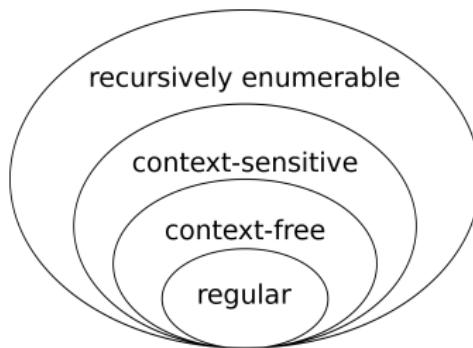


Figura 11: Clasificación de 1920 dada por Chomsky.

Clasificación de gramáticas (II)

Gramática	Lenguaje	Autómata	Reglas de producción
Tipo-0	Enumerable recursivamente	Máquina de Turing	$\gamma \rightarrow \alpha$ (sin restricciones)
Tipo-1	Sensible a contexto	Autómata linealmente acotado	$\alpha A \beta \rightarrow \alpha \gamma \beta$

Clasificación de gramáticas (III)

Gramática	Lenguaje	Autómata	Reglas de producción
Tipo-2	Libre de contexto	Autómata con pila	$A \rightarrow \alpha$
Tipo-3	Regular	Autómata de estados finitos	$A \rightarrow a,$ $A \rightarrow aB$

Sistemas de Lindenmayer

Las gramáticas L-system son gramáticas usadas para generar plantas y organismos.

$$\mathbf{G} = \{V, S, \omega, P\}$$

Donde:

- V = símbolos no terminales.
- S = símbolos terminales.
- $\omega \in V$ = define el estado inicial del sistema.
- P = conjunto de reglas de producción.

Ejemplo: números de Fibonacci

- $V = \{A, B\}$
- $S = \{\}$
- $\omega = \{A\}$
- $P = \{(A \rightarrow B), (B \rightarrow AB)\}$

Ejemplo de secuencia de Fibonnacci:

- A, B, AB, BAB, ABBAB, BABABBAB, ABBABBABABBAB,
BABABBABABBABBABABBAB, ...
- Las longitudes dan: 1, 1, 2, 3, 5, 8, 13, 21, ...

Ejemplo: curva de Koch (I)

- $V = \{F\}$
- $S = \{+, -\}$ (+ = giro izquierda 90°, - = giro derecha 90°)
- $\omega = \{F\}$ (F = dibujar 1 paso hacia delante)
- $P = \{(F \rightarrow F + F - F - F + F)\}$

Ejemplo de secuencia de curva de Koch:

- $F; F+F-F-F+F; F+F-F-F+F+F+F-F-F+F-F+F-F+F-F+F-F+F+F+F+F-F-F+F;$
 $F+F-F-F+F+F+F-F-F-F+F-F+F-F-F+F-F+F-F+F-F+F-F+F+F+F+F-F+F+$
 $F+F-F-F+F+F+F-F-F-F+F-F-F+F-F-F+F-F+F-F+F+F+F+F-F-F+F-$
 $F+F-F-F+F+F+F-F-F-F+F-F-F+F-F-F+F-F+F+F+F+F-F-F+F-$
 $F+F-F-F+F+F+F-F-F-F+F-F-F+F-F-F+F-F+F+F+F+F-F-F+F+$
 $F+F-F-F+F+F+F-F-F-F+F-F-F+F-F-F+F-F+F+F+F+F-F-F+F; \dots$

Ejemplo: curva de Koch (II)

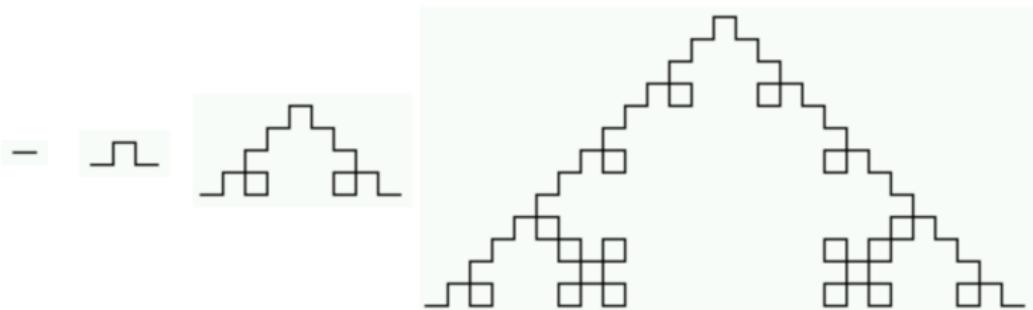


Figura 12: Curva de Koch generada con gramática L-system.

Ejemplo: árbol 2D

- $V = \{Y, T\}$ (Y = yema [mitad de tamaño que tallo], T = tallo)
- $S = \{[,]\}$ ($[$ = bifurcación a izquierda, $]$ = bifurcación a derecha)
- $\omega = \{Y\}$
- $P = \{(Y \rightarrow T[Y]Y), (T \rightarrow TT)\}$

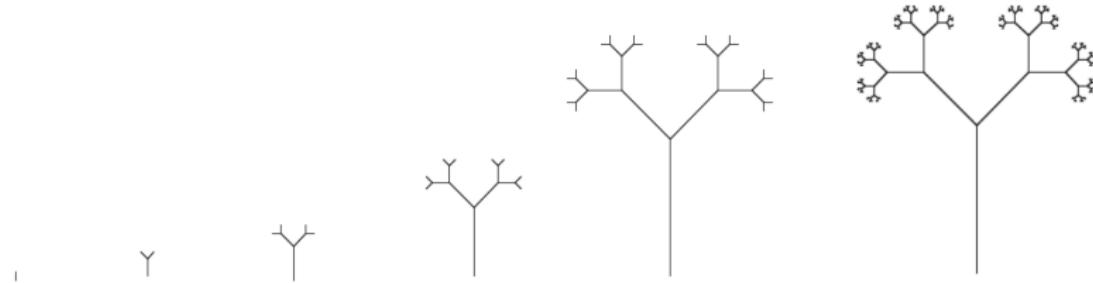


Figura 13: Ejemplo de gramática L-system para la generación de árbol 2D.

Aún más ejemplos

- Becker, S., Peter, M., Fritsch, D., Philipp, D., Baier, P., & Dibak, C. (2013). **Combined grammar for the modeling of building interiors.** ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci, 1-6.

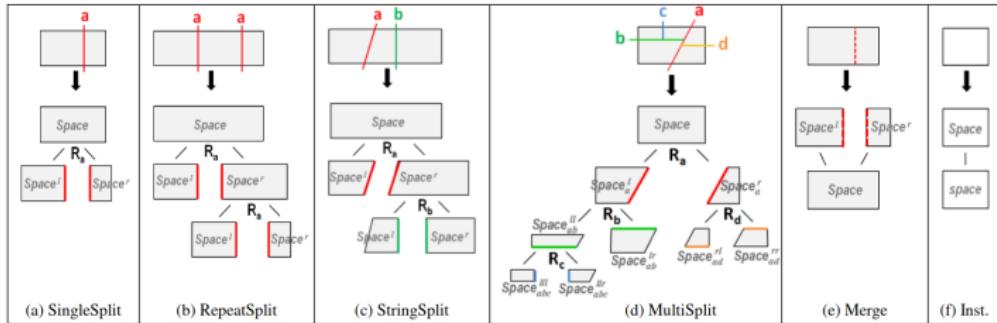


Figura 14: Generación de interiores de edificios con gramáticas L-system.

Tema 3 - Elementos (assets) y captura de datos.

3.2 Técnicas de desenrollado y texturas 2D.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Contenido del tema

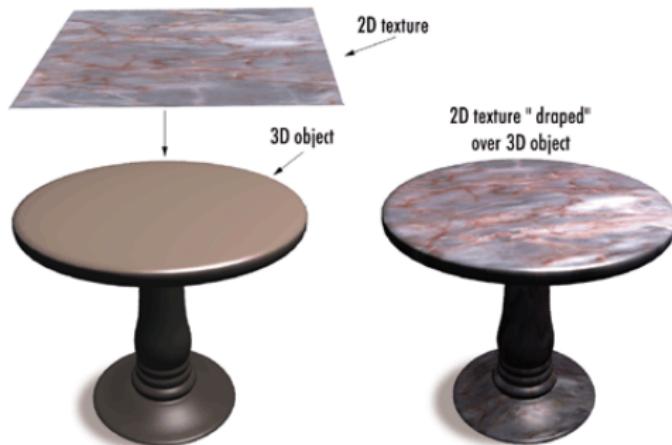
Tema 3: Elementos (assets) y captura de datos.

- 3.1 Nubes de puntos y capturas mediante escáneres 3D.
- 3.2 Técnicas de desenrollado y texturas 2D.
- 3.3 Simplificación de modelos 3D y texturización automática.
- 3.4 Materiales y shaders de iluminación.
- 3.5 Nuevas técnicas software y hardware para la generación

3.2 Técnicas de desenrollado y texturas 2D.

Una **textura** es una imagen que se proyecta sobre una geometría.

Transferida de disco (SSD)/memoria a GPU.

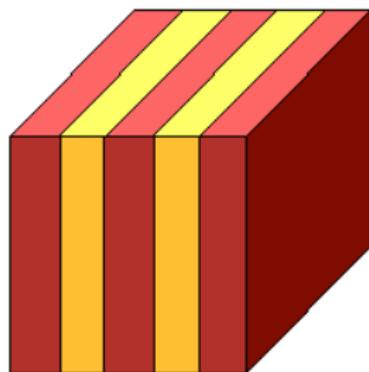


Tipos de texturas

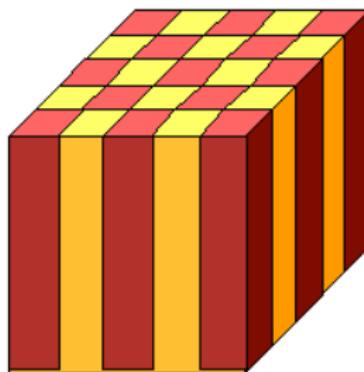
Una textura puede tener múltiples dimensiones: 1D, 2D o 3D.

La unidad básica de una textura es el **texel**.

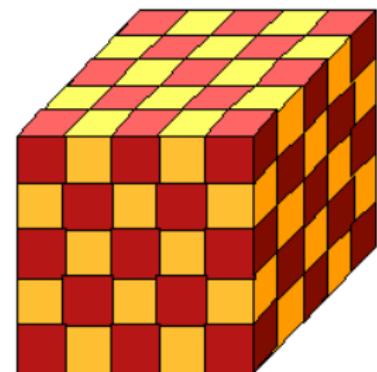
1-D



2-D



3-D



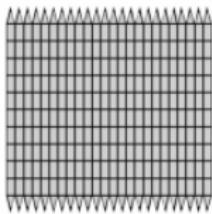
Texture mapping

La correspondencia se establece asociando al vértice las coordenadas del punto de la textura (coordenadas de textura). Las **coordenadas de textura** se dan **normalizadas**.

3-D Model



UV Map



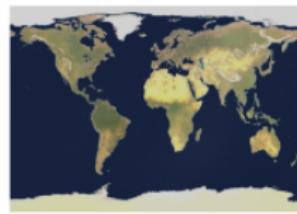
$$p = (x, y, z)$$



$$p = (u, v)$$



Texture



Parametrización (I)

Las coordenadas de textura se puede obtener desplegando las caras del objeto.

Problemas de realizar la parametrización:

- Islas (zonas no conexas).
- Aristas abiertas formando costuras (*seams*) en el modelo.

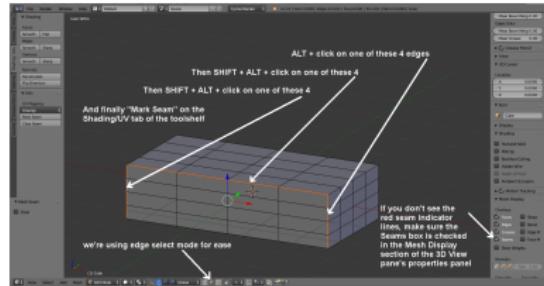


Figura 1: Proceso de marcado de costuras en Blender.

Parametrización (II)

Más problemas de realizar la parametrización:

- Zonas no utilizadas en la textura.

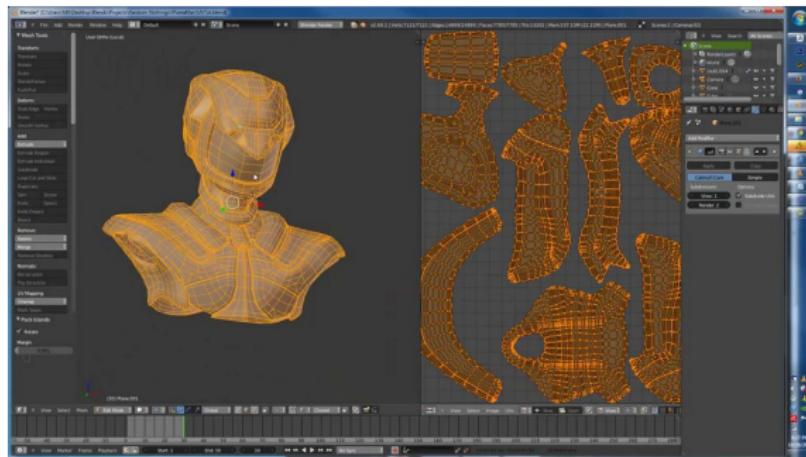


Figura 2: Ejemplo de textura no aprovechada al 100%.

Parametrización (III)

Tipos de parametrización:

- Completamente manual (triángulo a triángulo o por áreas).
- Completamente automática (aparecen problemas).
- Semi-automática (zonas desplegadas mediante algoritmos, marco de costuras, etc.)

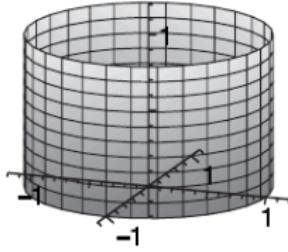
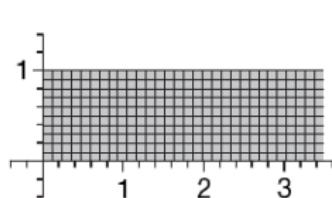
Parametrización automática (I)

Parametrización automática al crear un objeto:

- Envolver el objeto con una superficie paramétrica.
- Asignar coordenadas de textura en a partir de puntos correspondientes en la superficie.
- La parametrización es la inversa de la función paramétrica que define la superficie.

$$\text{parameterization: } f(u, v) = (\cos u, \sin u, v)$$

$$\text{inverse: } f^{-1}(x, y, z) = (\arccos x, z)$$



Parametrización automática (II)

Para cada vértice tendremos unas coordenadas UV asignadas.

Se pueden tener varios canales UV.

Las islas se consiguen duplicando los vértices.

```
var uvs = PoolVector2Array()  
...  
arr[Mesh.ARRAY_TEX_UV] = uvs
```

Parametrización automática (III)

La proyección genera distorsiones y no es unívoca.

Se puede obtener desplegando cintas de triángulos.

- Genera muchas costuras.
- Genera muchos espacios perdidos.
- Genera muchas islas.

Kai Hormann, Bruno Lévy, Alla Sheffer. **Mesh Parametrization: Theory and Practice.** Siggraph Course Notes, 2007.

Olga Sorkine and Daniel Cohen-Or. **Warped textures for UV mapping encoding.** SHORT PAPER EUROGRAPHICS 2001

Aplicaciones de la parametrización

Se utiliza entre otras operaciones para:

- *Morphing.*
- Reparar mallas.
- Materiales.
- Creación de terrenos.
- etc.

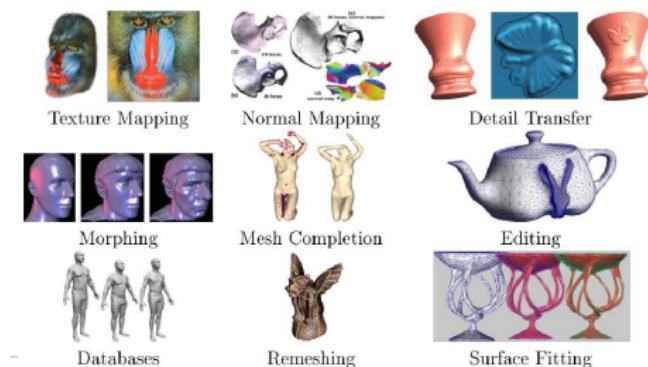


Figura 3: Usos comunes de las texturas.

Bump mapping (I)

Las texturas no solamente se utilizan para el color:

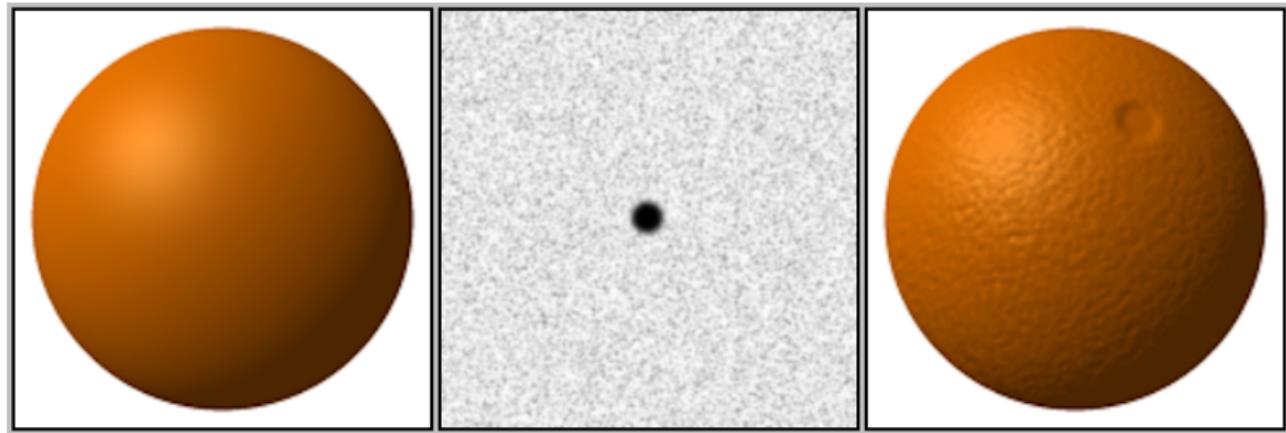


Figura 4: Ejemplo de bump mapping.

<https://upload.wikimedia.org/wikipedia/commons/9/93/FakeBump2D-animation.gif>

Bump mapping (II)

- ① Comprobar la altura del mapa que corresponde a la superficie.
- ② Calcular la normal a la superficie en el mapa de altura, típicamente mediante derivadas.
- ③ Combinar la normal de la superficie con la normal real (*geométrica*), combinándolas en una nueva dirección.
- ④ Calcular la interacción de la luz con la superficie con algún modelo de iluminación (ej. Lambert, Phong, etc.).

The Derivative

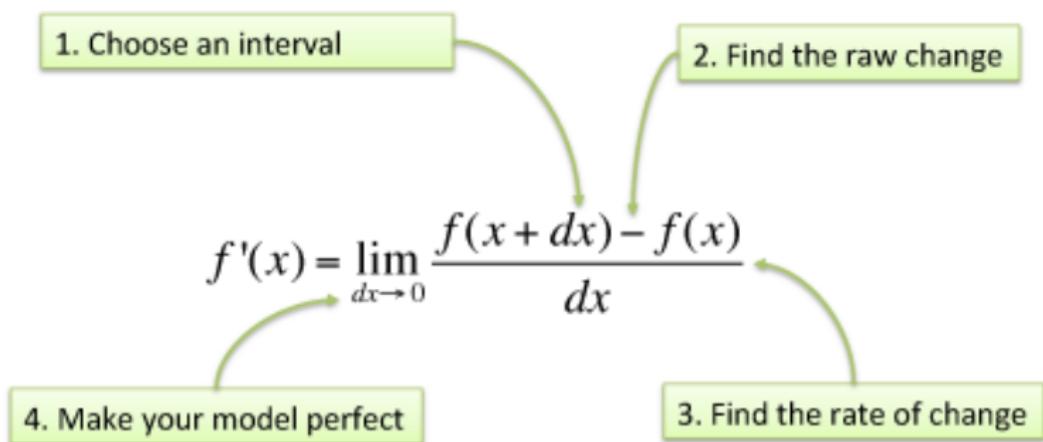


Figura 5: Definición de derivada.

Derivada de una imagen (II)

Convolución y *kernel*:

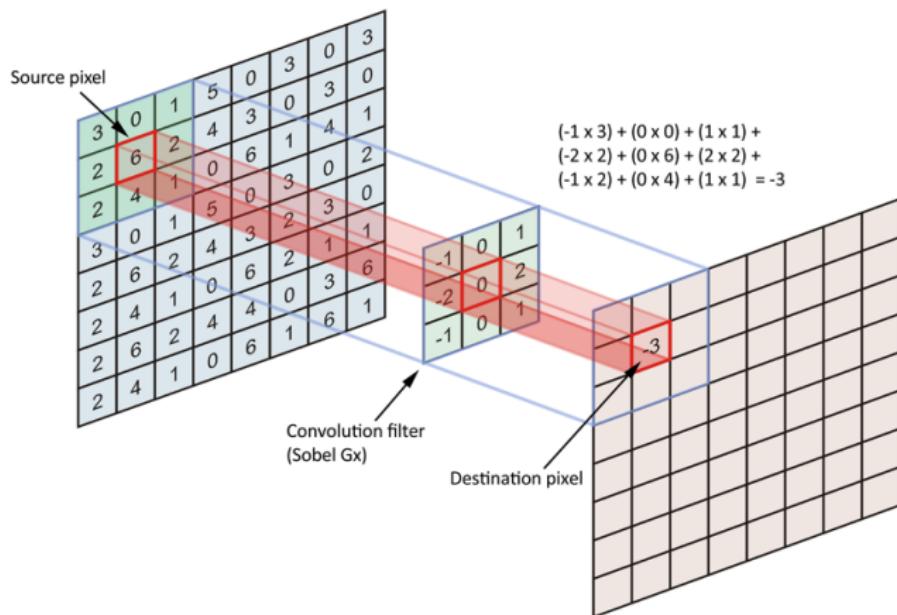


Figura 6: Operación de convolución.

Derivada de una imagen (III)

Kernel DoG:

Remember:
Derivative of Gaussian filter

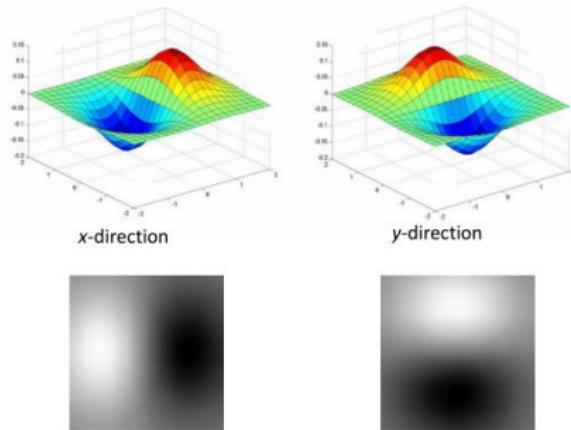


Figura 7: 1^a derivada de la Gaussiana.

Derivada de una imagen (IV)

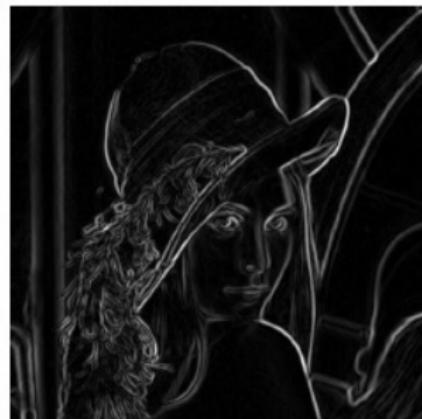
Magnitud y dirección:



X-Derivative of Gaussian



Y-Derivative of Gaussian



Gradient Magnitude

Figura 8: Direcciones (izq. y centro) y magnitud (derecha) de los vectores.

Normal mapping

Parecido al Bump Mapping, pero las normales del mapa ya vienen dadas en RGB: $\vec{n} = (n_x = R, n_y = G, n_z = B)$.

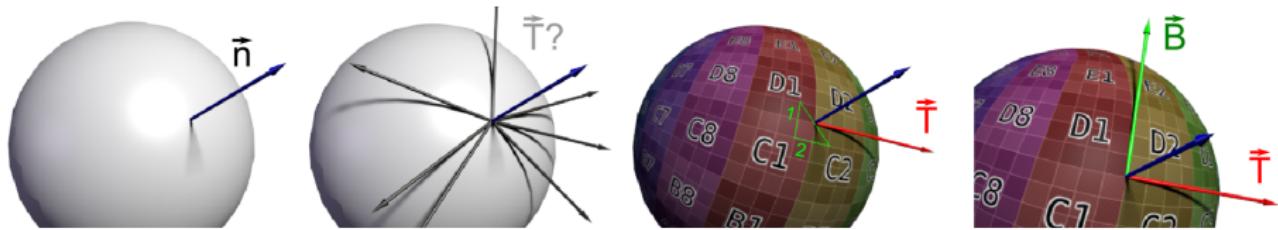


Figura 9: Ejemplo de imagen de normales.

Tangente y bitangente (I)

Calcular la tangente (\vec{T}) y la bitangente (\vec{B}) a partir de los lados del triángulo ($\Delta_1 pos$, $\Delta_2 pos$) usando sus coordenadas de textura ($\Delta_1 UV$, $\Delta_2 UV$):

- $\Delta_1 pos_x = \Delta_2 UV_x \cdot \vec{T} + \Delta_1 UV_y \cdot \vec{B}$
- $\Delta_2 pos_x = \Delta_2 UV_x \cdot \vec{T} + \Delta_2 UV_y \cdot \vec{B}$



Tangente y bitangente (II)

Teniendo \vec{T} , \vec{N} y \vec{B} pasamos todo al espacio de la tangente, usamos la inversa de la matriz TBN (descompuesta en filas), o (por eficiencia) su traspuesta:

$$\text{TBN}^T = \begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{pmatrix}$$

Solamente si primero la hacemos ortogonal. Para ello, hacemos la tangente perpendicular a la normal:

$$\vec{T} = \text{norm}([\vec{T} - \vec{N}] \cdot [\vec{N} \cdot \vec{T}])$$

The diagram shows three vectors originating from the same point: a vertical vector n pointing upwards, a horizontal vector t pointing to the right, and a diagonal vector $\vec{n} \cdot \vec{T}$ pointing down and to the right. A dotted line represents the projection of $\vec{n} \cdot \vec{T}$ onto the horizontal plane. A vector labeled $\text{dot}(n,t)$ points vertically downwards along this dotted line. A vector labeled $-n \cdot \text{dot}(n,t)$ points horizontally to the left, perpendicular to the direction of $\text{dot}(n,t)$. This vector $-n \cdot \text{dot}(n,t)$ is added to $\vec{n} \cdot \vec{T}$ to form the tangent vector \vec{T} .

Cocinado (bake) de texturas

Necesario que ambos modelos comparten el mismo espacio UV.



Figura 10: Modelo a alta resolución (izq.), modelo a baja (dcha.) y mismo modelo con normales precocinadas.

Tema 3 - Elementos (assets) y captura de datos.

3.5 Nuevas técnicas software y hardware para la generación de terrenos y su optimización.

Germán Arroyo, Juan Carlos Torres

3 de mayo de 2021

Contenido del tema

Tema 3: Elementos (assets) y captura de datos.

- 3.1 Nubes de puntos y capturas mediante escáneres 3D.
- 3.2 Técnicas de desenrollado y texturas 2D.
- 3.3 Simplificación de modelos 3D y texturización automática.
- 3.4 Materiales y shaders de iluminación.
- 3.5 Nuevas técnicas software y hardware para la generación

3.5 Nuevas técnicas software y hardware para la generación de terrenos y su optimización.



Figura 1: Terreno procedural: Siggraph 2013.

Textura 3D y volúmenes

Una textura 3D contiene información en el interior.

Puede ser usada para generar colores de altura en el terreno muy fácilmente.

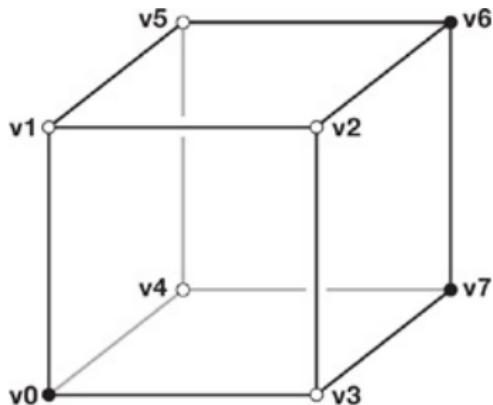


Figura 2: Terreno coloreado con una textura 3D.

Marching cubes (I)

Cada esquina tiene un signo: From each one we make a bit.

- Si la densidad es negativa el bit valdrá 0; si es positiva 1.



$$\begin{aligned}\text{Case} &= v7|v6|v5|v4|v3|v2|v1|v0 \\ &= 11000001 \\ &= 193\end{aligned}$$

Figura 3: Representación de conectividad con 8 véxoles.

Marching cubes (II)

Cada código obtenido está en rango [0, 255].

0 ó 255 significa que está fuera del terreno. En otro caso genera triángulos.

- Se busca en una tabla de casos del 1 al 254 (normalmente en GPU) y se generan los triángulos (vértices en alguno de los doce lados).
- Se interpolan, el vértice debería caer donde la densidad es aproximadamente 0.

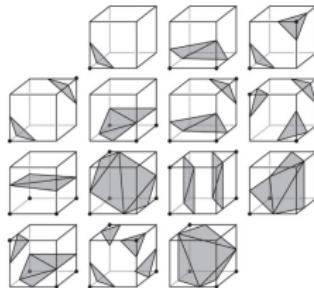


Figura 4: Algunos casos de marching cubes.

Marching cubes (IV)

Nvidia tiene un muy buen tutorial para generar terrenos cambiando solamente la frecuencia de la textura 3D (podemos usar Perlin Noise).

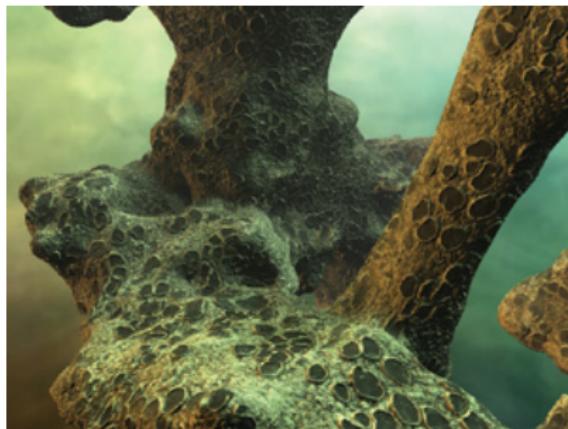


Figura 5: Terreno generado tres proyecciones de planares.

<https://developer.nvidia.com/gpugems/gpugems3/part-i-geometry/chapter-1-generating-complex-procedural-terrains-using-gpu>

Level of Detail (LOD)

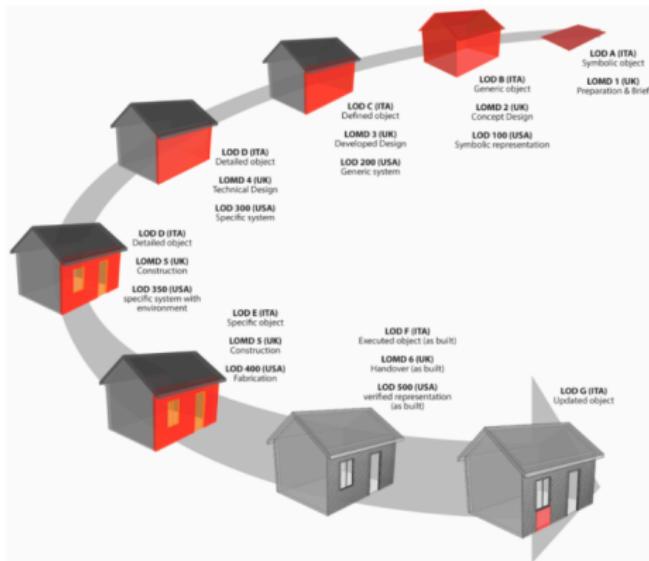


Figura 6: Ejemplo de Level of Detail.

Billboards e impostores

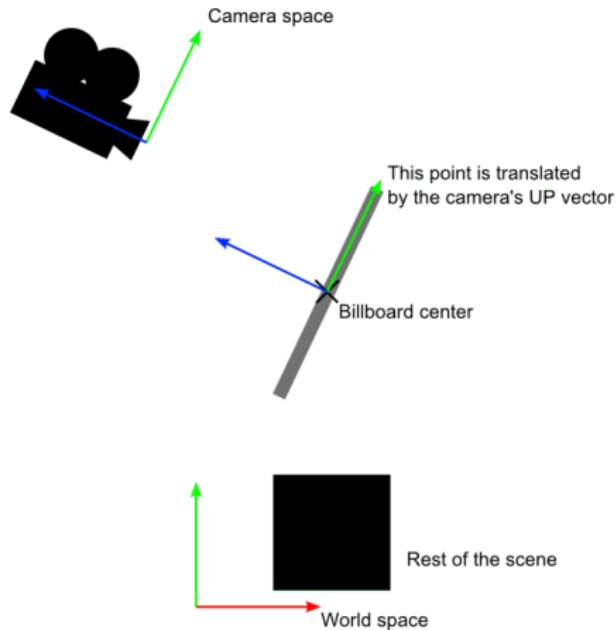


Figura 7: Billboard.

Tema 4 - Interacción y sistemas de visualización en RV y RA.

4.3 Sensores software. Sistemas de eventos y scripting.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Contenido del tema

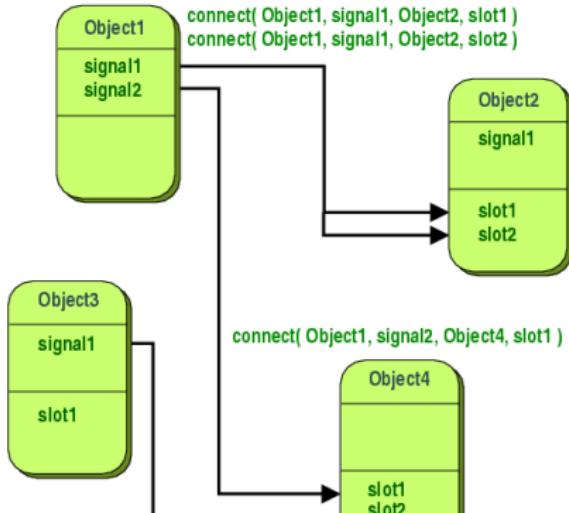
Tema 4: Interacción y sistemas de visualización en RV y RA

- 4.1 Realidad virtual y percepción visual.
- 4.2 Dispositivos de visualización e interacción.
- 4.3 Sensores software. Sistemas de eventos y scripting.
- 4.4 Sistemas de Realidad Aumentada.

4.3 Sensores software. Sistemas de eventos y scripting.

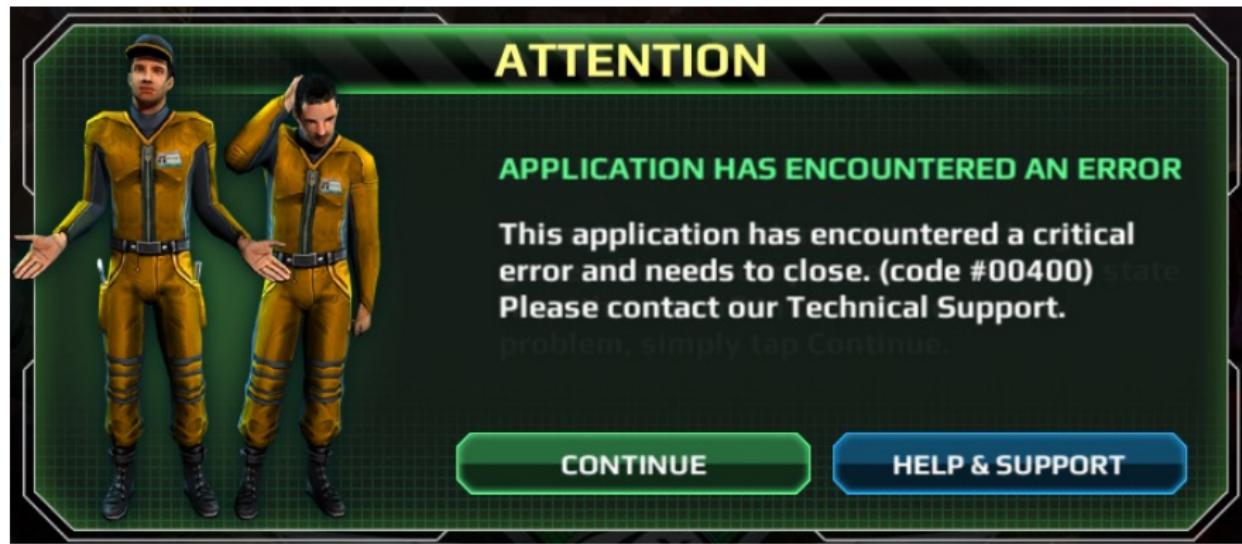
Técnicas de programación de sensores software (línea de tiempo, colisiones, fuerzas, UI, AI, etc.):

- Extensión (herencia): Button → mi_boton
- Callbacks (funciones): int event(void *obj_data, ...);
- Señales.



Problema de la sincronización

Un simple «if» en el momento inadecuado puede romper un mundo virtual. Necesitamos sincronizar todas las partes (física, visualización, interacción, etc.).



Necesitamos un *observador* que vigile que todo se ejecuta en el orden

Patrón de diseño: Basado en Observador

El **Observador** (*observer*) es un objeto internamente síncrono (aunque externamente asíncrono) que acepta peticiones de los **sujetos**, las ordena y las ejecuta en el momento correspondiente.

Una vez hecha la petición, el **sujeto** no detiene su ejecución, sino que continúa con sus instrucciones.

- Si necesita esperar, puede usar semáforos.

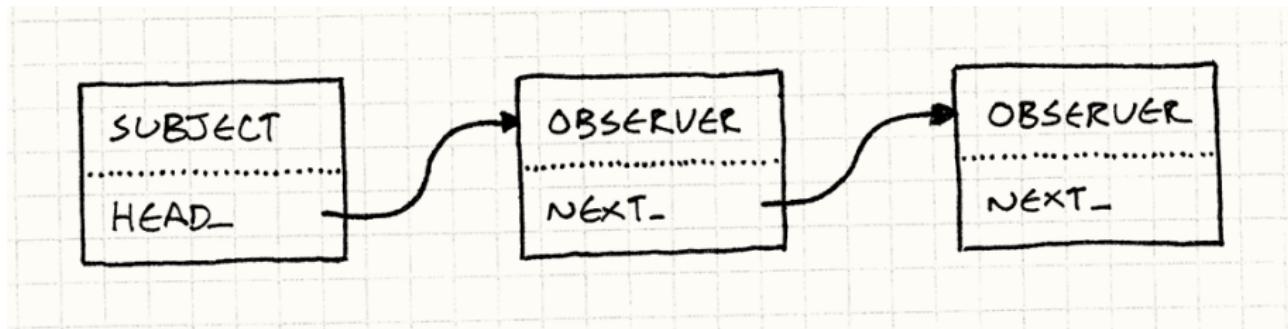
¡Problema!

- ¡Si un observador intenta cerrar el cerrojo que tiene un sujeto, el mundo virtual puede entrar en un bloqueo (*deadlock*)!

Observadores enlazados

En lugar de que un único observador se encargue de todas las partes, es común tener cada observador dedicado a un aspecto: física, visualización, etc.

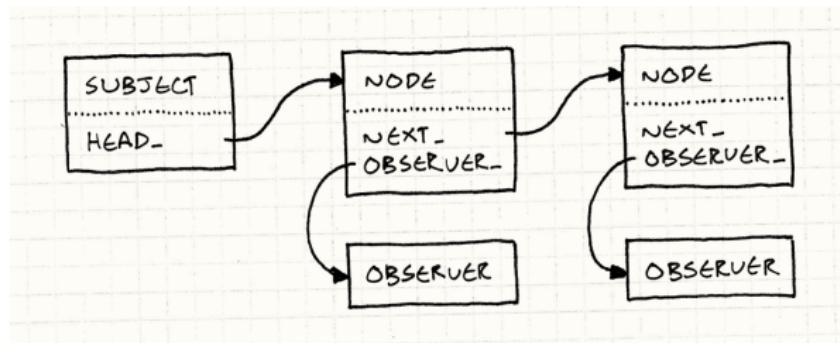
Los sujetos mantienen una lista de observadores (conectados bidireccionalmente).



Pool de observadores

Un observador podría estar en más de una de estas listas enlazadas.

- Evitar duplicidad: se crea un *pool* de objetos en su lugar.



Problemas de los observadores

- Aplicar observadores al problema equivocado: interfaces de usuario.
- Borrar observadores: aún puede haber sujetos que lo invoquen.
 - ▶ ¿Solución? Colector de basura: cada vez que hay un evento especial para un único objeto, se crea y se destruye (¿o no? → ¡se añade otra instancia!).

The *lapsed listener problem*:

https://en.wikipedia.org/wiki/Lapsed_listener_problem

¿Entonces?

- Simplemente se les notifica cuando algo ha cambiado.
- Lenguaje imperativo para modificar algo de la interfaz para reflejar un nuevo **estado**.

Señales

Las señales son una forma de emitir mensajes desde un objeto, mensaje con el que otros objetos pueden reaccionar.

Definición (emisor, solamente 1):

```
signal health_depleted
```

Conexión (emisor, solamente 1):

```
character_node.connect(«health_depleted», self, »_on_Character_health_d
```

Recepción (receptores, más de uno):

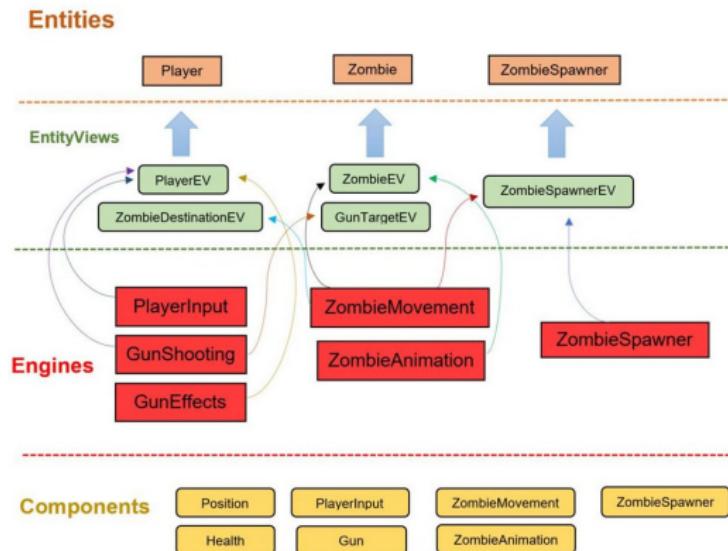
```
func _on_Character_health_depleted():
```

```
...
```

Entity Component System (ECS)

El ECS da preferencia a la composición, antes que a la herencia de propiedades.

Otorga una flexibilidad enorme, ya que cualquier cosa puede ser una **entidad**.



Componentes de un ECS

- **Entidad:** la entidad es un objeto de propósito general de identidad única (ID único).
- **Componente:** los datos del objeto (desligados de la entidad) para un aspecto de la entidad (etiquetas de comportamiento).
- **Sistema:** cada sistema se ejecuta constantemente, realizando las acciones globales para cada entidad que posea un determinado tipo de componente.

El código se escribe en los sistemas y no en las entidades o en los componentes.

Ejemplo de ECS

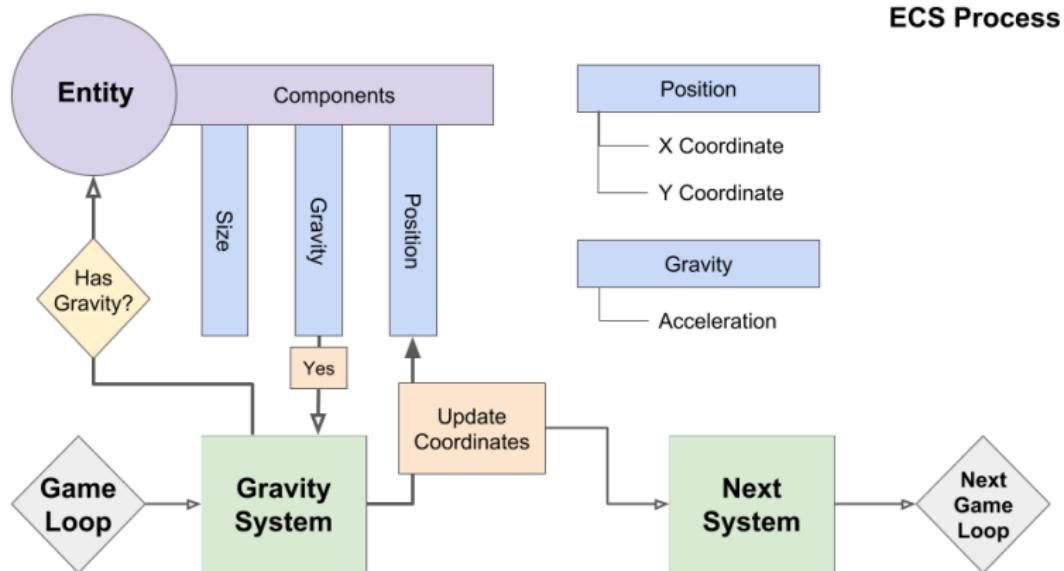


Figura 2: Ejemplo de Entity-Component-System en un entorno 2D.

Ejemplo de sistemas en un ECS

Los sistemas están enlazados, recordando la arquitectura del **Observador**.

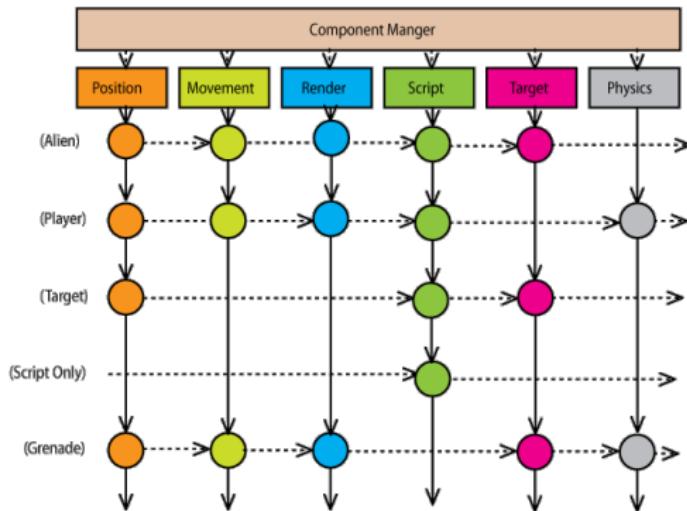


Figure 2 Object composition using components, viewed as a grid.

Figura 3: Ejemplo de conexión síncrona de sistemas.

Implementación mediante grupos (I)

A menudo los grupos funcionan como componentes. Son etiquetas asociadas a un nodo (que hace de entidad).

En el objeto que hace de entidad guardia:

```
add_to_group(«guards»)
```

Opcional: en algún evento podemos conectar etiqueta con *callback*:

```
get_tree().call_group(«guards», «enter_alert_mode»)
```

Implementación mediante grupos (II)

Dos opciones:

- ① El mismo objeto hace de sistema (usando callback anterior):

```
func enter_alert_mode():
```

```
...
```

- ② Otro objeto hace de sistema:

```
var guards = get_tree().get_nodes_in_group(«guards»)
```

Modo herramienta (*tool mode*)

Los scripts no se suelen ejecutar dentro del editor.

- Solo las propiedades exportadas pueden modificarse.

```
export (int) var gravity
```

¿Y si queremos que se ejecute dentro del propio editor?

```
tool
```

Tema 5 - Física y colisiones. Efectos especiales.

5.1 Introducción a los motores físicos.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

Contenido del tema

Tema 5: Física y colisiones. Efectos especiales.

- 5.1 Introducción a los motores físicos.
- 5.2 Interacción con dispositivos de entrada y dispositivos de salida.
- 5.3 Técnicas de optimización.
- 5.4 Personalización de fuerzas
- 5.5 Efectos especiales y técnicas volumétricas.
- 5.6 Shaders de vértices y técnicas avanzadas.

5.1 Introducción a los motores físicos.

La simulación física trata de reproducir el comportamiento dinámico y cinemático de los objetos de la escena.

Implica:

- Representar el estado de los objetos: posición, velocidad, aceleración y momento angular.
- Representar propiedades físicas de los objetos: densidad, elasticidad, coeficiente de fricción.
- Resolver las ecuaciones de la mecánica del sistema (integración en el tiempo).

Componentes

Detección de colisiones:

- Cinemática.
- Cálculo de velocidades.
- Dinámica.
- Cálculo de fuerzas.
- Fractura.

Objetos

- Puntual.
- Rígido.
- Deformable.
- Fluido.
- Tela.

What is Time Integration ?

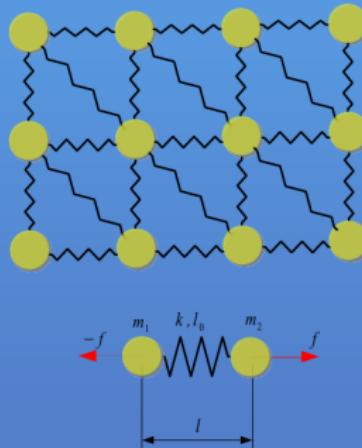
- Computing the simulation state at the next time step $t+h$ given :
 - State at the current time t
 - positions \mathbf{x}_t
 - velocities \mathbf{v}_t
 - Equations and constraints expressing the required mechanics
 - $\mathbf{a} = a(\mathbf{x}, \mathbf{v}, t)$
 - $c(x, v, t) \geq 0$
- Main issue : lively but stable simulation

$$\begin{aligned}\mathbf{v}(t) &= \mathbf{v}(0) + \int_0^t \mathbf{a}(\mathbf{x}, \mathbf{v}, t) dt \\ \mathbf{x}(t) &= \mathbf{x}(0) + \int_0^t \mathbf{v}(t) dt\end{aligned}$$

Métodos de simulación física (II)

Example : Springs

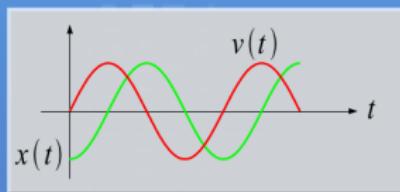
- Particle
 - mass m ,
 - position, velocity
- Spring
 - rest length l_0
 - stiffness k
- Force
 - $l = (x_2 - x_1)$
 - $f = k(l - l_0)$
- Acceleration
 - $a = f / m$



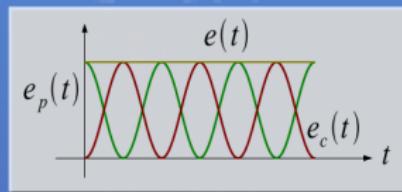
Métodos de simulación física (III)

Simplest Case

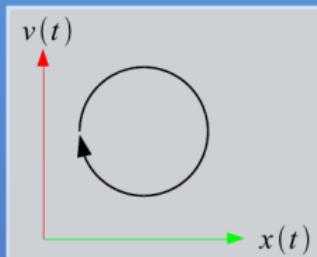
- Single 1D Particle
- Theoretical solution :



- Conservation of Energy :

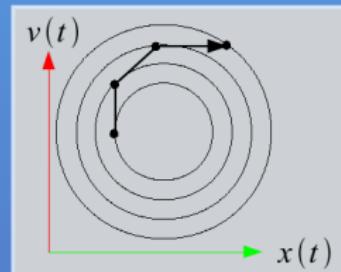


in phase space :



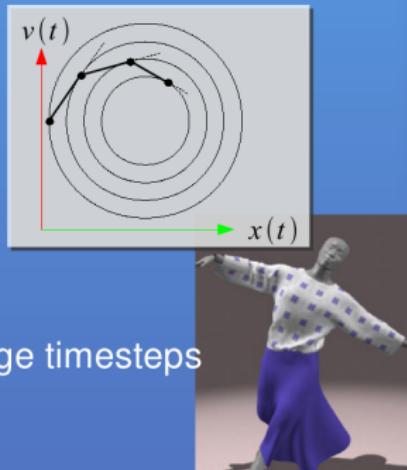
Explicit Integration

- Principle : use velocity and acceleration at the **begin** of the time-step
- Forward Euler :
$$\mathbf{a}_t = \mathbf{M}^{-1}\mathbf{f}$$
$$\mathbf{x}_{t+h} = \mathbf{x}_t + h\mathbf{v}_t$$
$$\mathbf{v}_{t+h} = \mathbf{v}_t + h\mathbf{a}_t$$
- Simple
- Exaggerates motion
- Damping is required
- Stiff systems require very small time steps



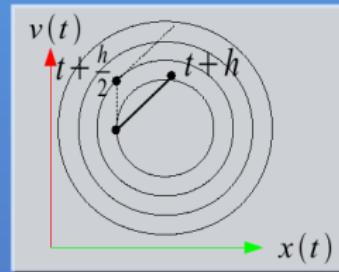
Implicit Integration

- Principle : use velocity and acceleration at the **end** of the time-step
- Backward Euler :
 - Solve $(M - \frac{1}{h} \frac{\partial f}{\partial v} - \frac{1}{h^2} \frac{\partial f}{\partial x}) a = (f + h \frac{\partial f}{\partial x} v)$
 - $v_{t+h} = v_t + h a_{t+h}$
 - $x_{t+h} = x_t + h v_{t+h}$
- Complex (requires a solver)
- Under-estimates motion
- Introduces additional damping
- Stable stiff systems even with large timesteps
- Well-suited for soft bodies



Explicit Runge-Kutta Methods

- 2nd-order Runge-Kutta (RK2) :
 - Go to $t + h/2$ using forward Euler
 - Compute the derivative
 - Use this derivative in a full forward Euler step
- Simple
- More precise than forward Euler
- Still exaggerates motion
- Well-suited for rigid



Colisiones en sistemas dinámicos (I)

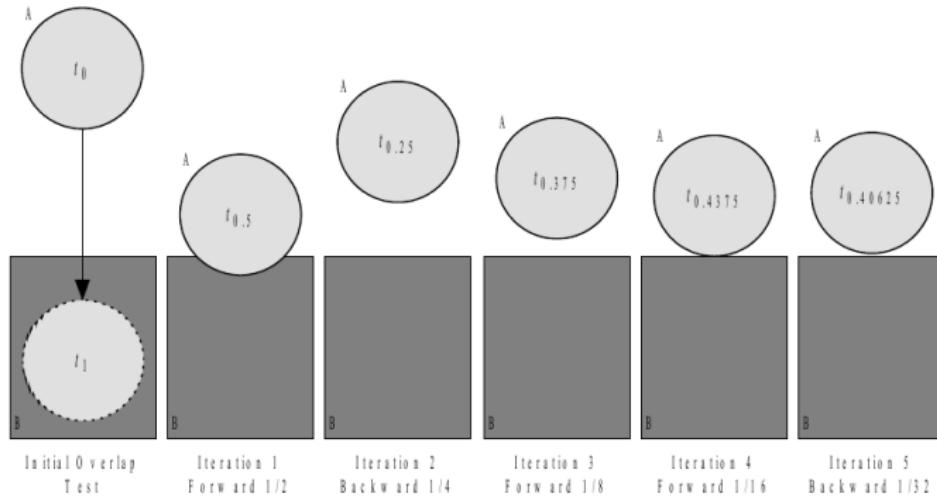
Escenas dinámicas: Si al menos uno de los objetos está en movimiento, a partir de sus posiciones iniciales y finales.

- El objetivo es determinar el punto de contacto.
- En simulación necesitamos además saber el tiempo de la colisión.

Colisiones en sistemas dinámicos (II)

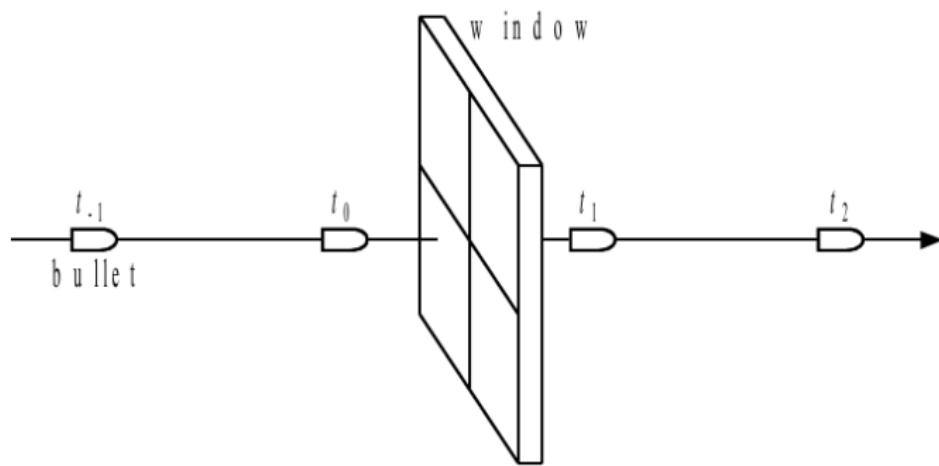
Tiempo de colisión: Se puede calcular retrocediendo en el tiempo hasta el momento de colisión.

- Se pueden usar técnicas de bisección.



Colisiones en sistemas dinámicos (III)

Cálculo de colisiones: Determinar superposición puede hacer que no se detecten colisiones si el tiempo de integración es muy alto.



La simulación es costosa:

- **Tiempo real.** Cálculo aproximado: Juegos, Entornos virtuales, Sistemas interactivos.
- **Precisos (lentos).** Películas, Aplicaciones científicas y técnicas.

Motores físicos

Open source:

- ODE.
- NEWTON.
- Bullet.

Comerciales:

- Havok (Intel).
- Physx (nVidia).
- Vortex (Montreal).
- Realflow (Nextlimit, España).

Tema 5 - Física y colisiones. Efectos especiales.

5.3 Técnicas de optimización.

Germán Arroyo, Juan Carlos Torres

18 de mayo de 2021

Contenido del tema

Tema 5: Física y colisiones. Efectos especiales.

- 5.1 Introducción a los motores físicos.
- 5.2 Interacción con dispositivos de entrada y dispositivos de salida.
- 5.3 Técnicas de optimización.
- 5.4 Personalización de fuerzas
- 5.5 Efectos especiales y técnicas volumétricas.
- 5.6 Shaders de vértices y técnicas avanzadas.

5.3 Técnicas de optimización.

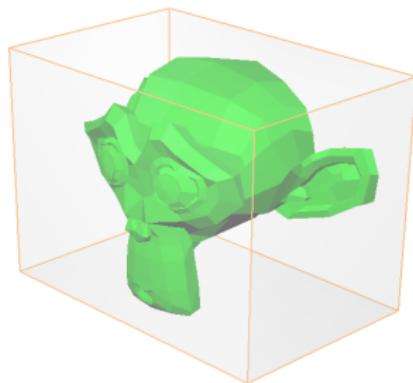
Técnicas de aceleración:

- Funciones cuyo coste depende del tamaño del modelo:
 - ▶ Visualización.
 - ▶ Selección.
 - ▶ Calculo de colisiones.
 - ▶ Sensores.
 - ▶ Inteligencia artificial.
 - ▶ etc. etc.

Volumenes envolventes (I)

Axis-Aligned-Bounding-Box (AABB):

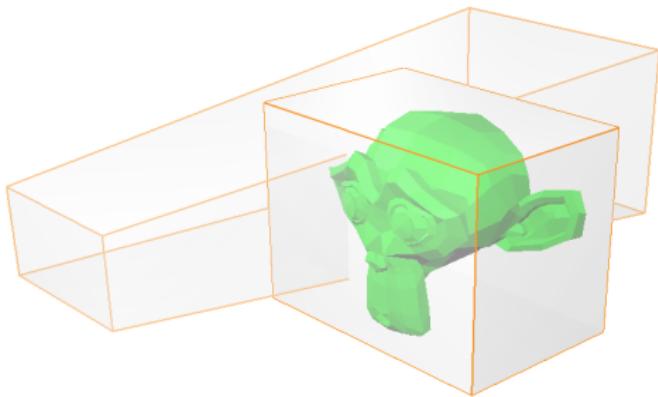
- El más simple.
- ① Recorrer vértices.
 - ② Obtener $A = (x_{min}, y_{min}, z_{min})$, $B = (x_{max}, y_{max}, z_{max})$.



Volumenes envolventes (II)

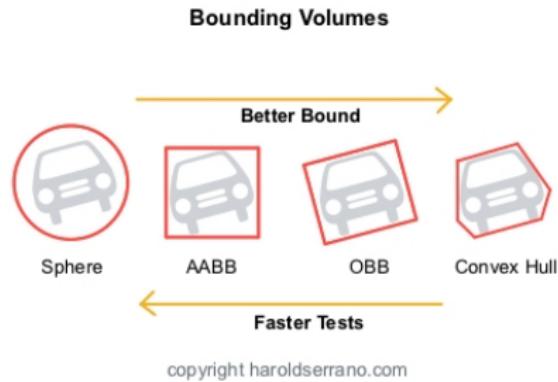
Eliminación de partes no visibles:

- Si la envolvente es visible: Visualizar componente.



Volumenes envolventes (III)

Física: La efectividad dependerá de lo ajustado que sea el volumen. - El coste de cálculo, almacenamiento y filtrado también.



<https://www.haroldsserrano.com/blog/tips-for-developing-a-collision-detection-system>

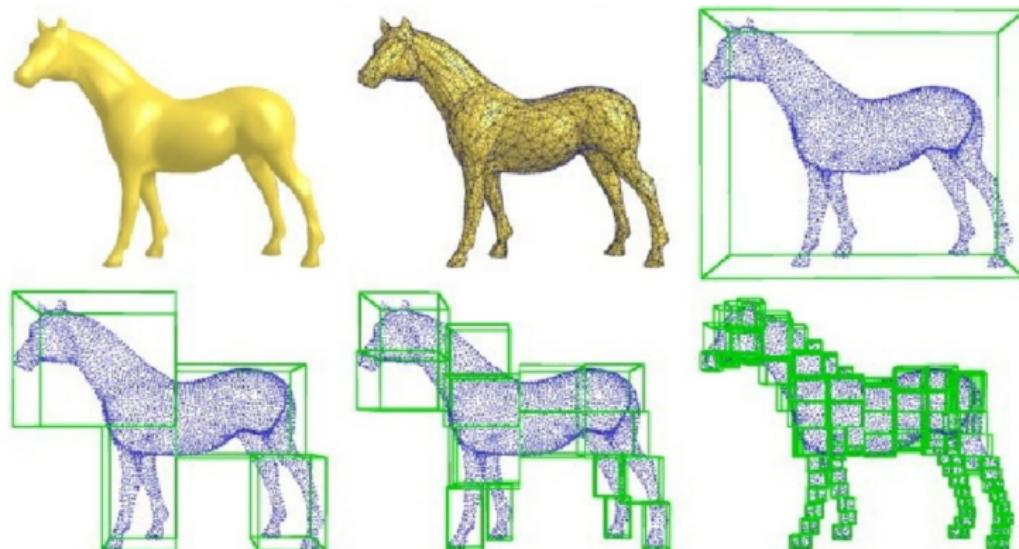
Índexación espacial

¿Cómo visualizar entornos enormes?



Volumenes envolventes jerárquicos

J.P. Sauta, D.Sidobrebc: **Efficient models for grasp planning with a multi-fingered hand** Author links open overlay panel.

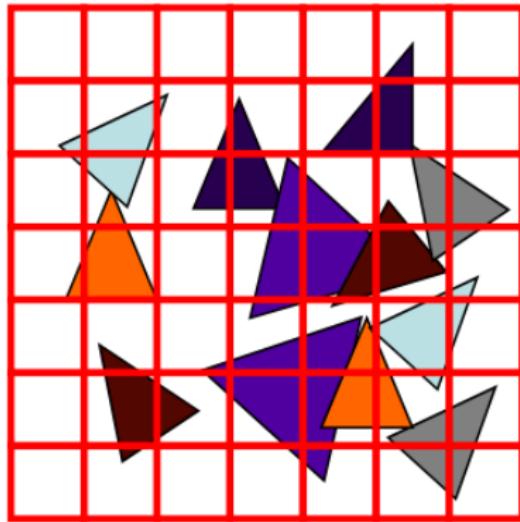


[https:](https://www.sciencedirect.com/science/article/abs/pii/S0921889011001515)

[//www.sciencedirect.com/science/article/abs/pii/S0921889011001515](https://www.sciencedirect.com/science/article/abs/pii/S0921889011001515)

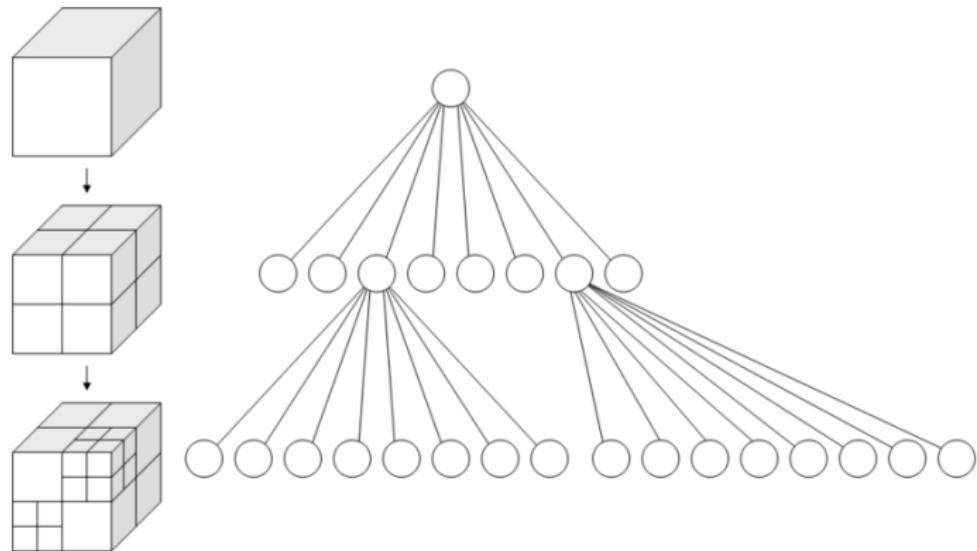
Indexación: Rejilla

- Se subdivide el espacio en celdas.
- En cada celda se almacena la lista de objetos que contiene.
- El índice puede ser un array 3D o un *hash*.



Indexación: Octree

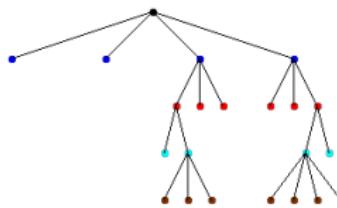
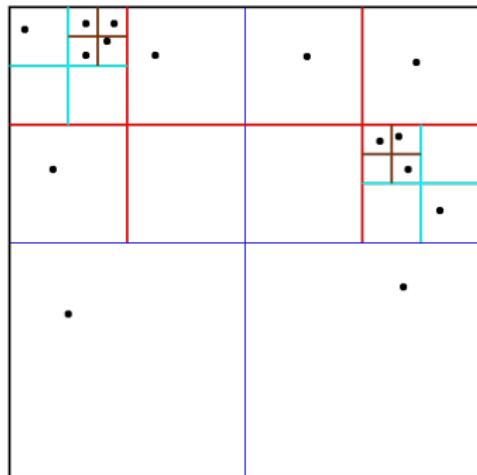
- El nodo raíz representa el cubo envolvente de la escena.
- Cada nodo se subdivide en 8 subnodos (hasta un tamaño o un número de objetos contenidos prefijado).
- Los nodos terminales almacenan la lista de objetos.



Indexación: Quadtree

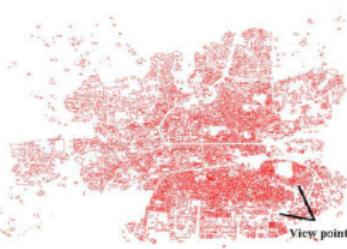
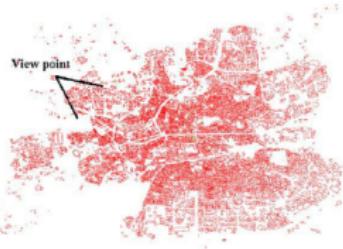
- Igual que un octree pero en el plano.
- Se puede utilizar para indexar un plano del entorno.

Adaptive quadtree where no square contains more than 1 particle



Visualización adaptativa

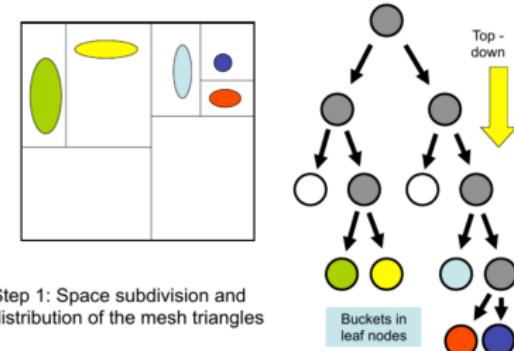
Se seleccionan los componentes de la escena y su nivel de detalle en función del punto de vista.



Ejemplo de visualización adaptativa: Impostores (I)

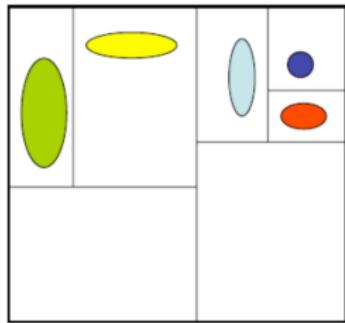
C. Andujar, P. Brunet, A. Chica, I. Navazo, **Visualization of Large-Scale Urban Models through Multi-Level Relief Impostors**, Computer Graphics Forum, Volume 29 (2010), number 8 pp. 2456–2468. 2010.

- 1 Generación de un índice espacial.

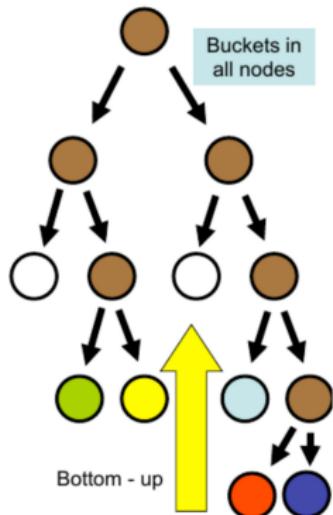


Ejemplo de visualización adaptativa: Impostores (II)

- ② Generación de niveles de detalle intermedios en cada nodo.



Step 2: Simplification: information all tree nodes



Ejemplo de visualización adaptativa: Impostores (III)

- ③ Calcula la profundidad de cada rama en el árbol.

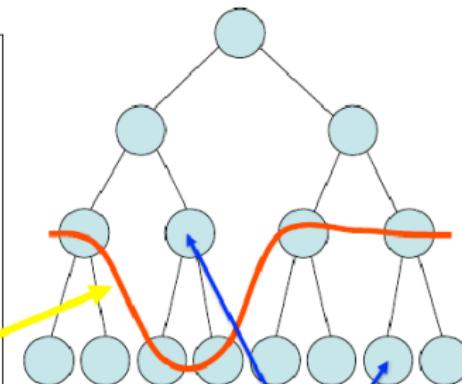
Visualization of:

- Triangle meshes
- Point clouds
- Impostors, IBR

Scene tree

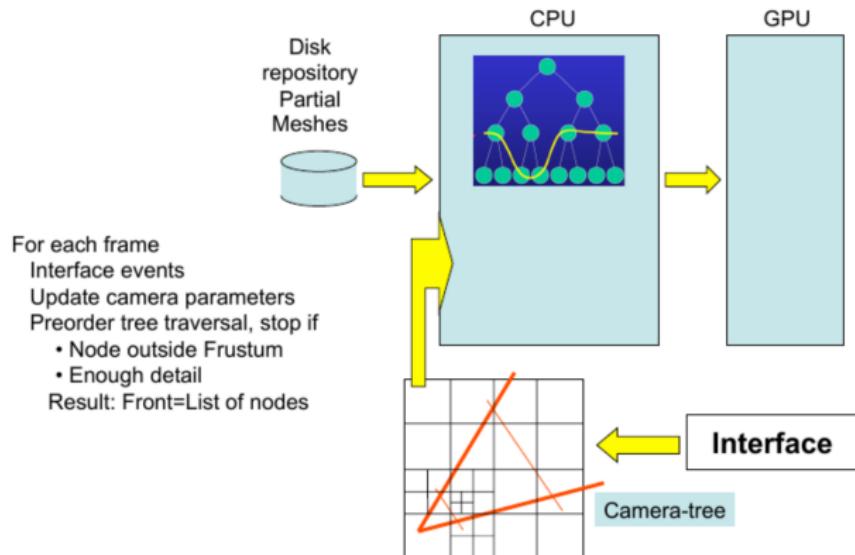
- **Construction**
- Simplification
- Tree of clusters of primitives
- Tree in main RAM memory
- Disk Repository: Data
- **Rendering**
- View-dependent Front
- Cache: on CPU and GPU

Quality metric required
Time critical rendering,
Rendering in a Budget



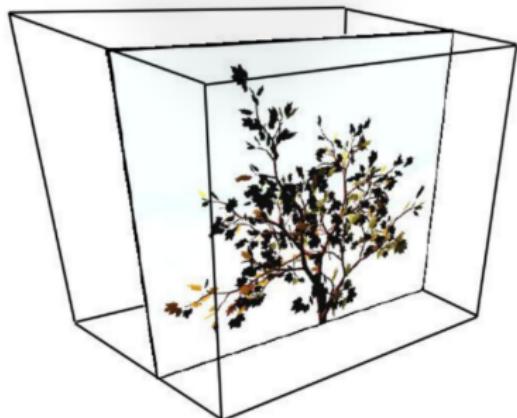
Ejemplo de visualización adaptativa: Impostores (IV)

- ④ El frente se actualiza en cada frame según la posición de cámara.



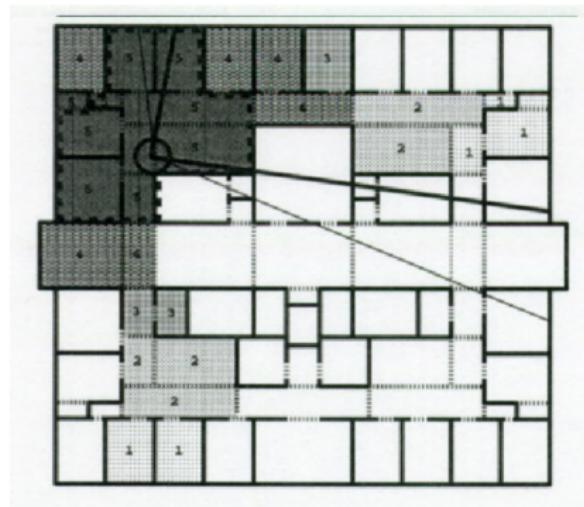
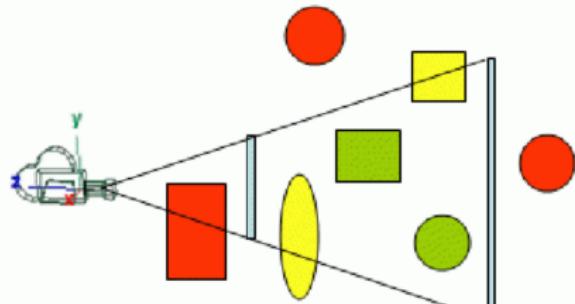
Ejemplo de visualización adaptativa: Impostores (V)

- Se sustituye una geometría compleja por una imagen prerenderizada.
- La imagen se debe sustituir si el punto de vista cambia de forma apreciable.



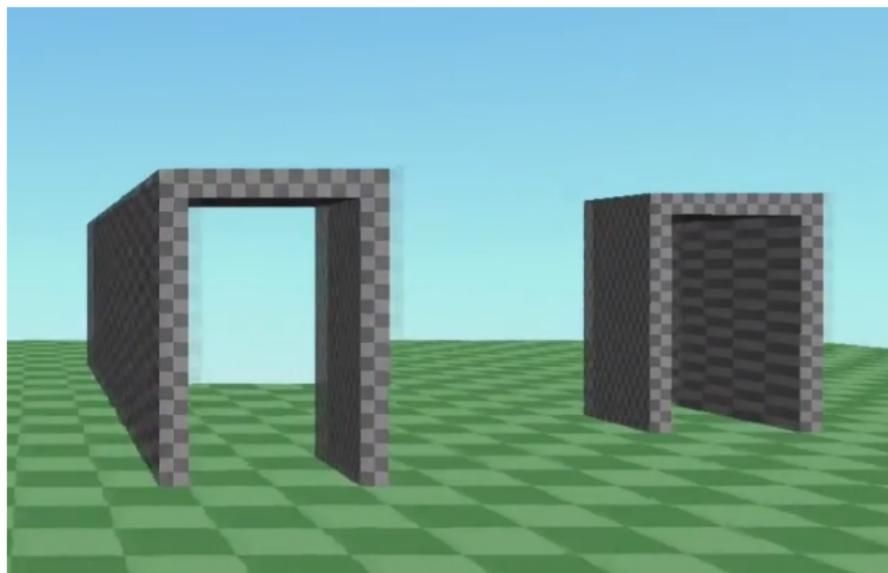
Portales (I)

Para escenarios formados por estancias conectadas se puede precalcular la visibilidad, almacenando en cada una los elementos contenidos y las estancias que son visibles desde ella.



Portales (II)

FBO + grafos de conectividad = salas.



<https://youtu.be/JAy-0c89KJY?t=219>

Portales (III)



Figura 1: Fotograma del juego *Portal*.

Tema 5 - Física y colisiones. Efectos especiales.

5.5 Efectos especiales y técnicas volumétricas.

Germán Arroyo, Juan Carlos Torres

5 de febrero de 2021

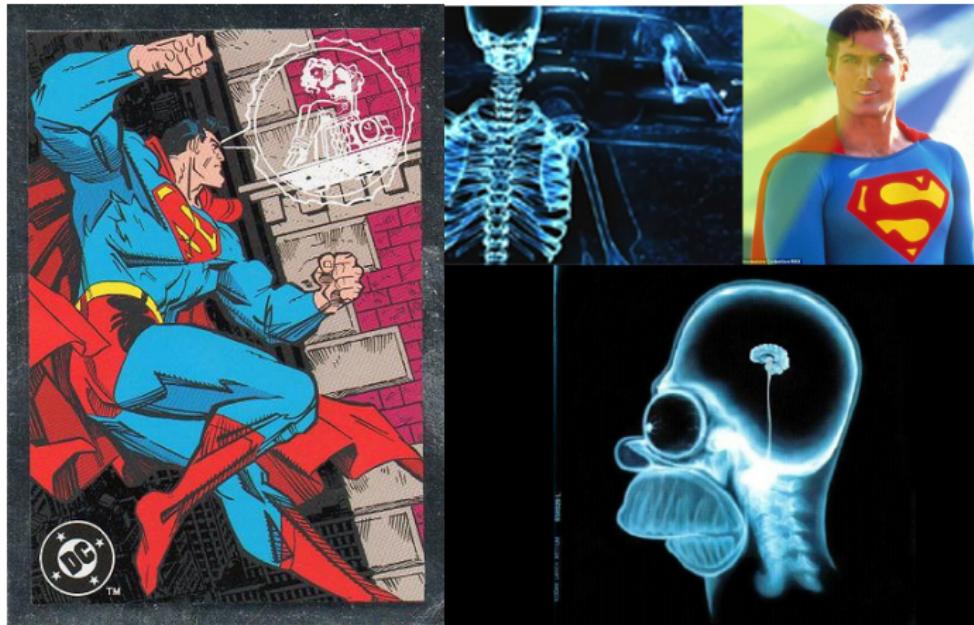
Contenido del tema

Tema 5: Física y colisiones. Efectos especiales.

- 5.1 Introducción a los motores físicos.
- 5.2 Interacción con dispositivos de entrada y dispositivos de salida.
- 5.3 Técnicas de optimización.
- 5.4 Personalización de fuerzas
- 5.5 Efectos especiales y técnicas volumétricas.
- 5.6 Shaders de vértices y técnicas avanzadas.

5.5 Efectos especiales y técnicas volumétricas.

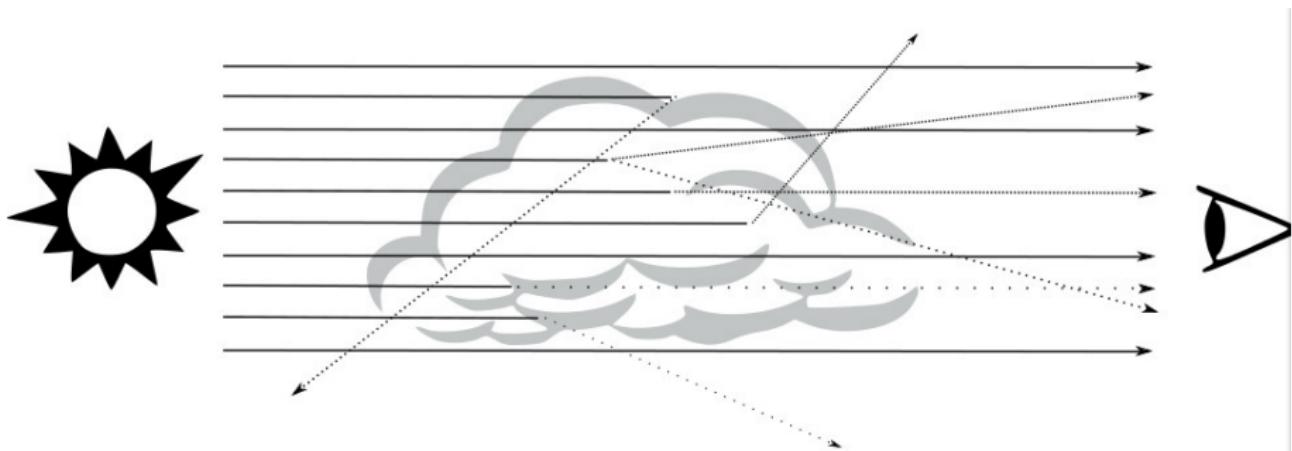
Visualización directa:



Fundamentos (I)

Pérdida de energía (extinción):

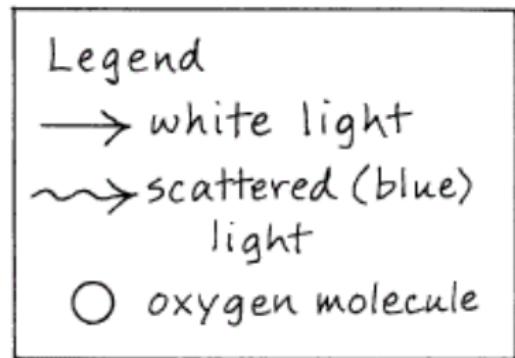
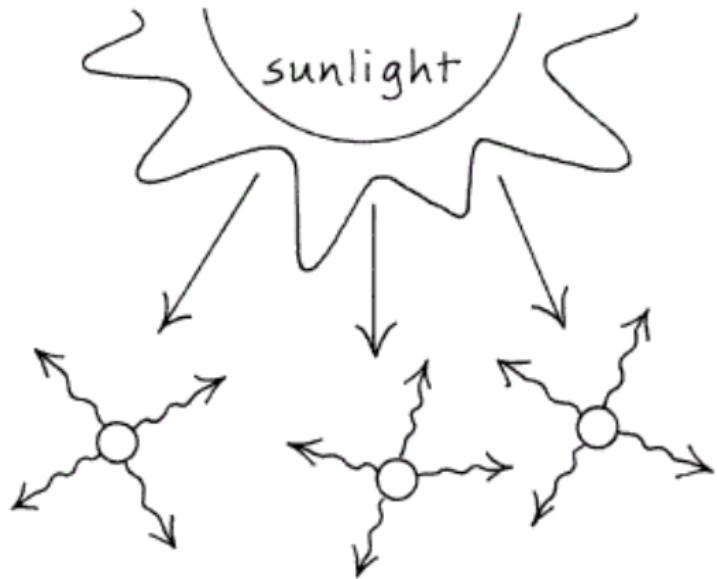
- La luz puede absorberse (líneas de puntos)
- La luz puede dispersarse por las partículas.



- $\vec{r}(s) = \vec{p} + s\hat{d}$

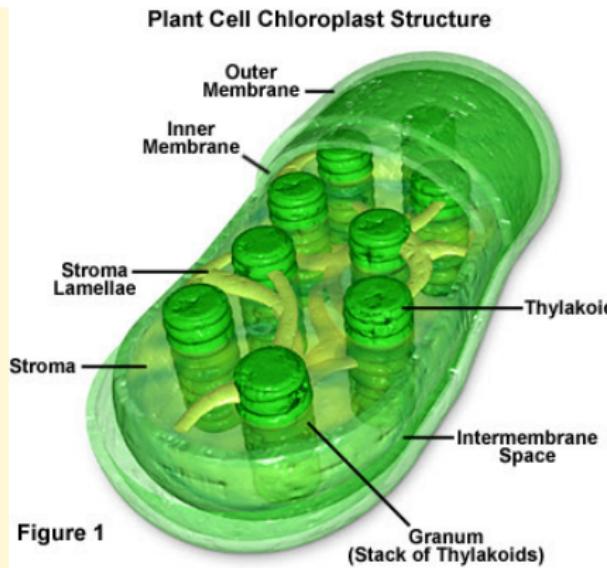
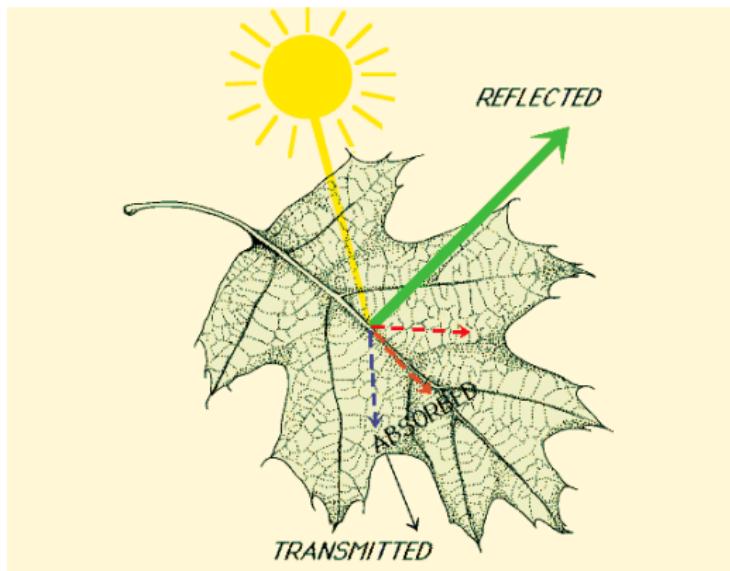
Fundamentos (II)

Ejemplo dispersión: cielo azul.



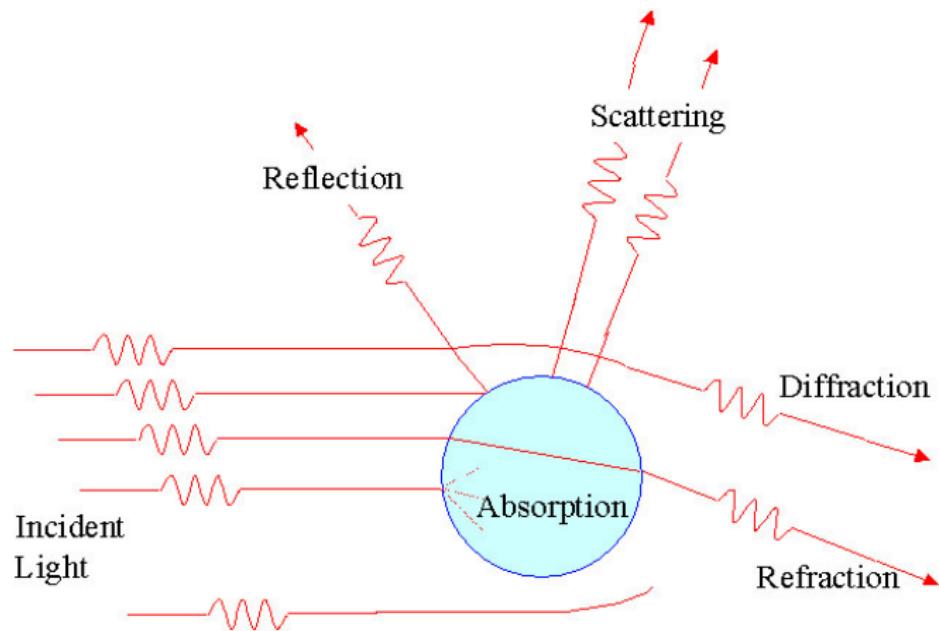
Fundamentos (III)

Ejemplo absorción: clorofila.



Fundamentos (IV)

Modelo de iluminación más realista.



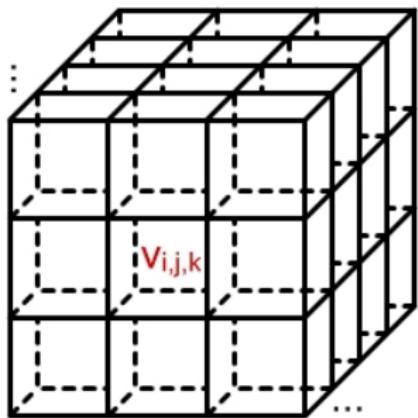
Estructura de datos (I)

Voxel: representación en intersección y en celda.

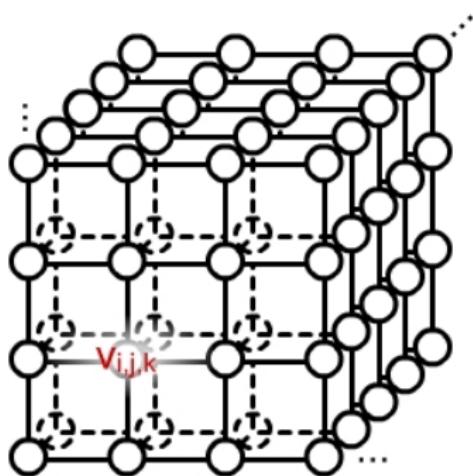
A: Typical Voxel



B: Voxel Set

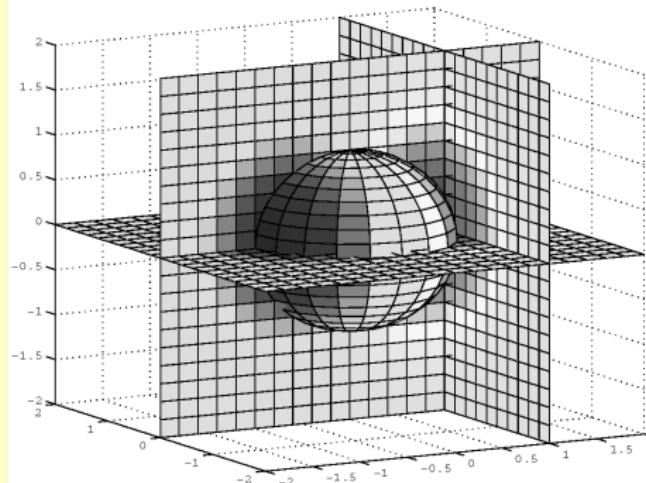
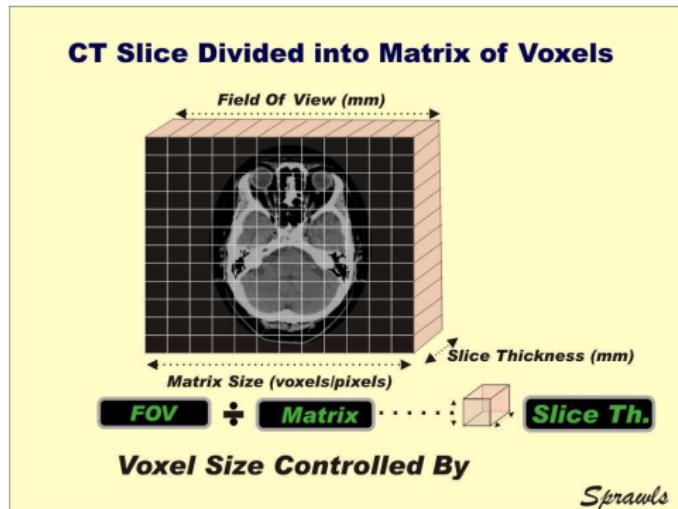


C: Voxel Grid



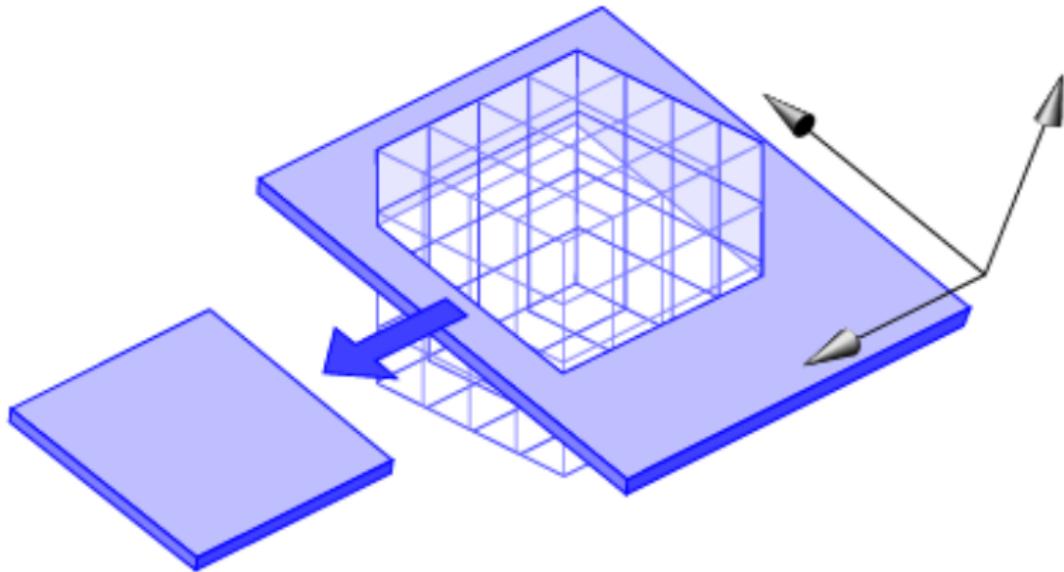
Estructura de datos (II)

Adquisición.

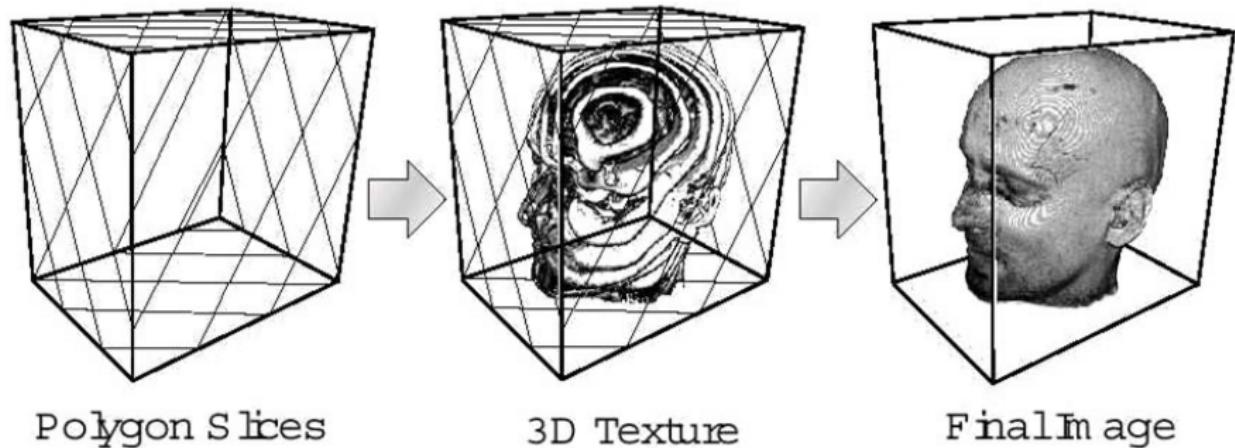


Estructura de datos (III)

Texturas 3D.

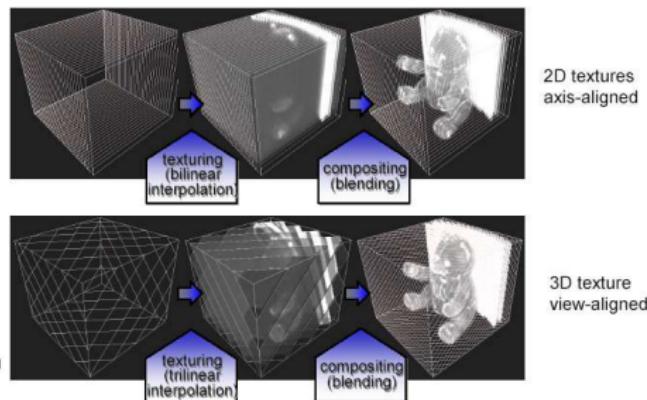
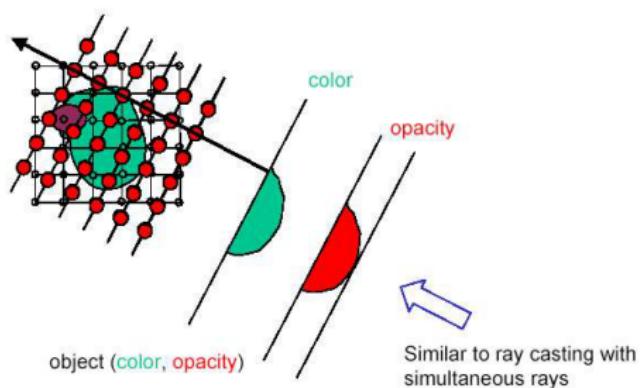


Rotación de texturas



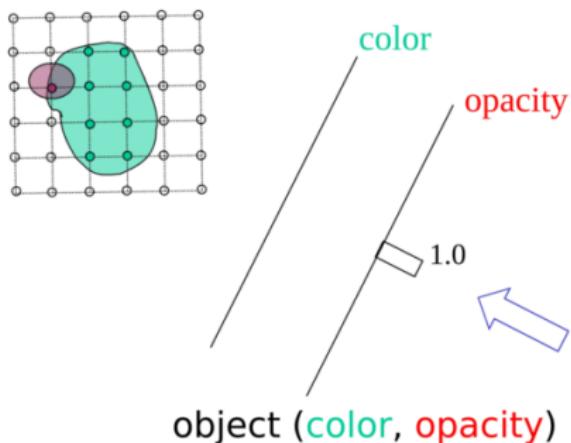
Composición de imagen final

Interpolación trilineal.

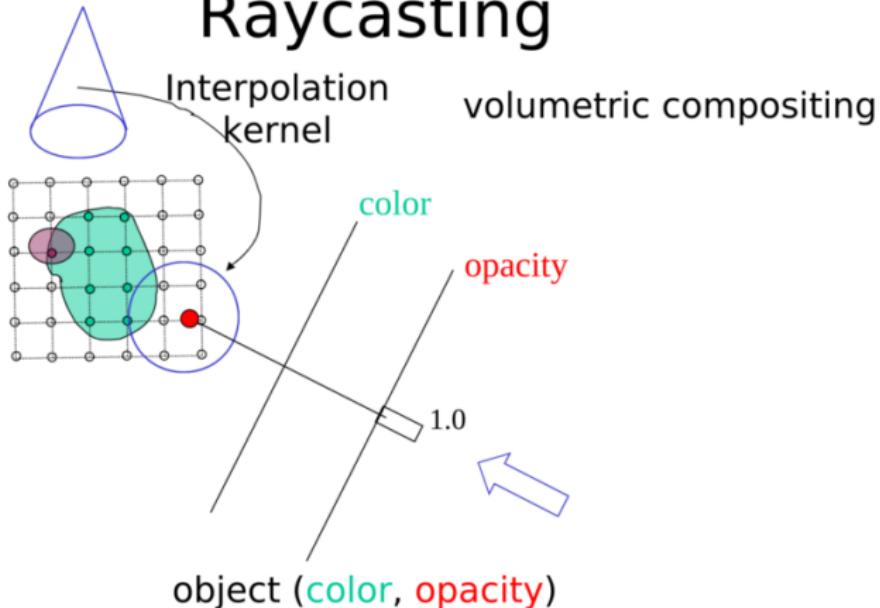


Raycasting

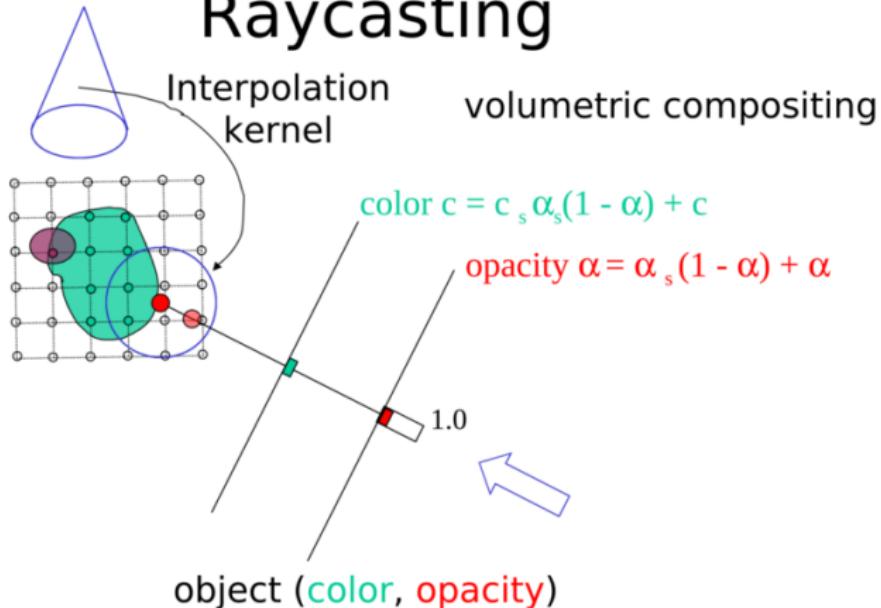
volumetric compositing



Raycasting

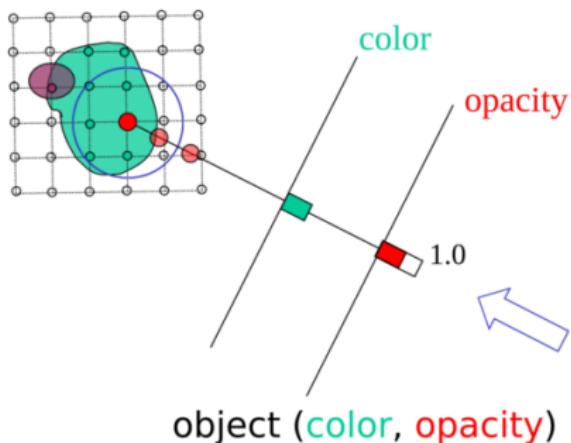


Raycasting



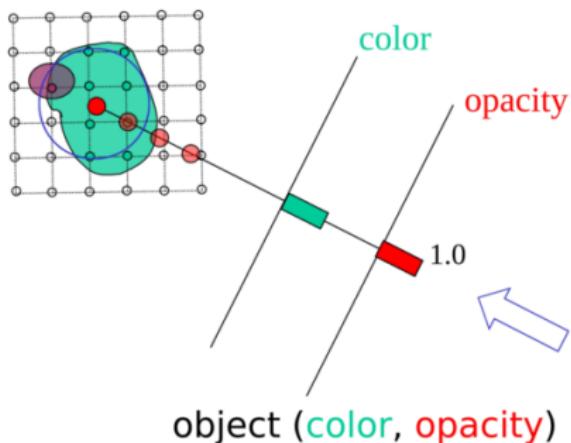
Raycasting

volumetric compositing



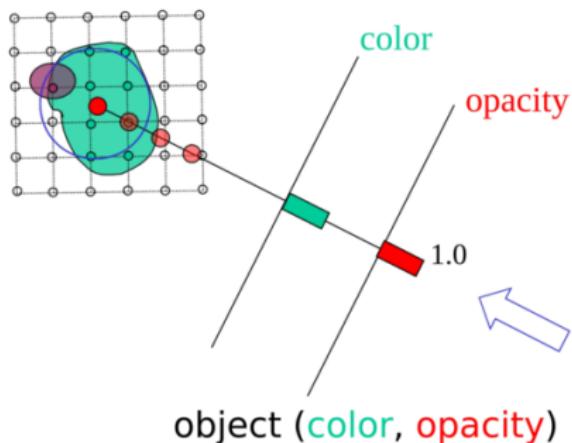
Raycasting

volumetric compositing



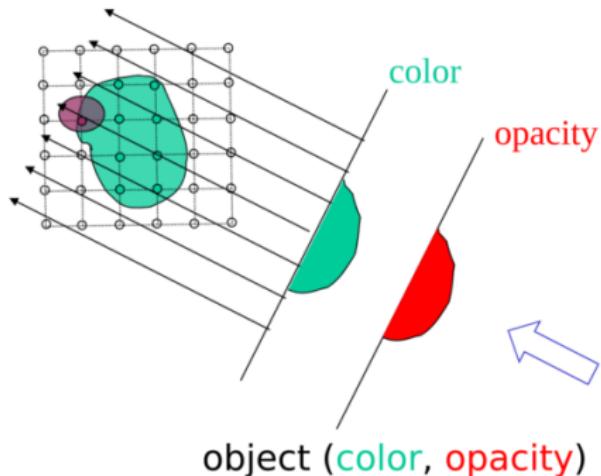
Raycasting

volumetric compositing



Raycasting

volumetric compositing



Iluminación global

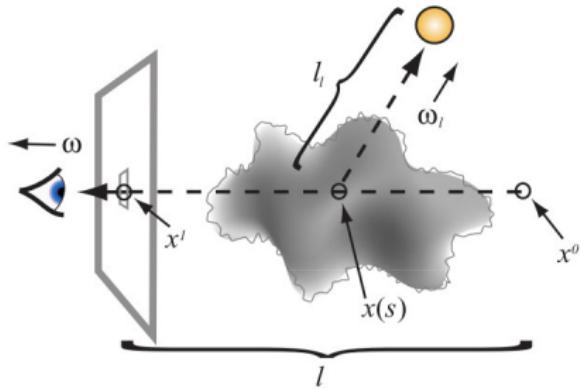


Fig. 2. Geometric setup used in volume shading equations.

Symbol	Definition
\mathbf{x}	Generic location
s	Distance from the ray's origin
$\mathbf{x}(s)$	Generic location along a ray
R	Surface reflectance
E	Emission in a volume
$T(s, l)$	Attenuation along the ray from $\mathbf{x}(s)$ to $\mathbf{x}(l)$
$\vec{\omega}$	Generic direction
$\vec{\omega}_l$	The light direction
τ	Attenuation coefficient
L_l	Point light source intensity
$L_l(s)$	Light intensity at point $\mathbf{x}(s)$
P	Phase function
l	Generic ray length
l_l	Light ray length

Ejemplos (I)

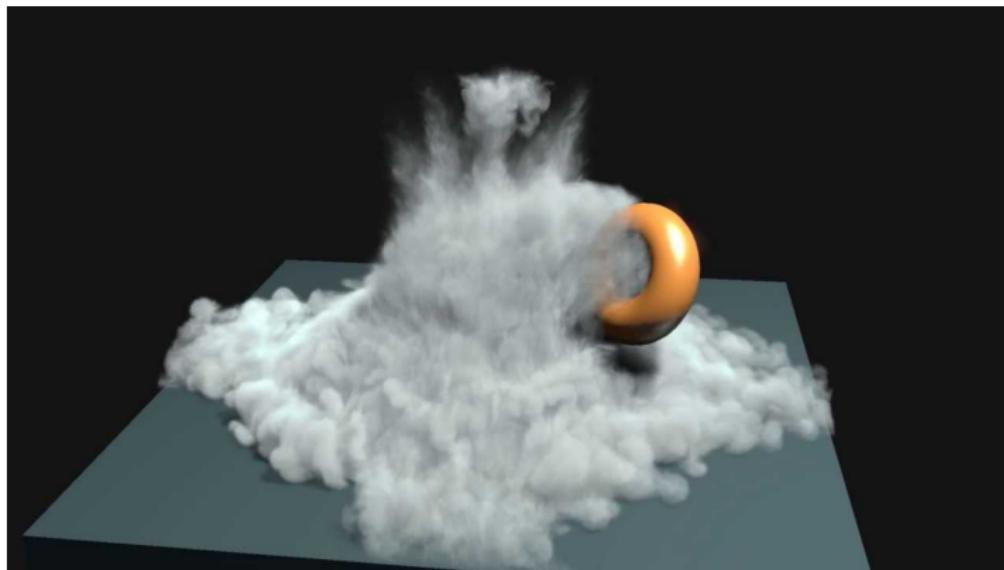
Nubes:



<https://pdfs.semanticscholar.org/5690/0e49e56db385e711ad622b1df17b006cb37c.pdf>

Ejemplos (II)

Humo, líquidos, etc.



Tema 5 - Física y colisiones. Efectos especiales.

5.6 Shaders de vértices y técnicas avanzadas.

Germán Arroyo, Juan Carlos Torres

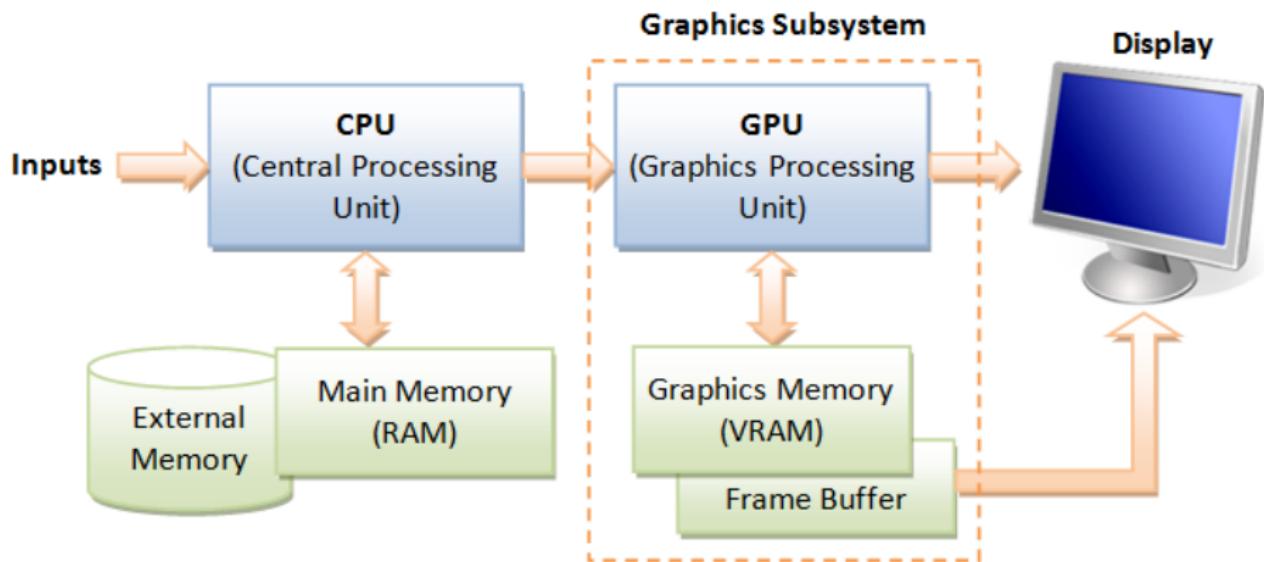
20 de mayo de 2021

Contenido del tema

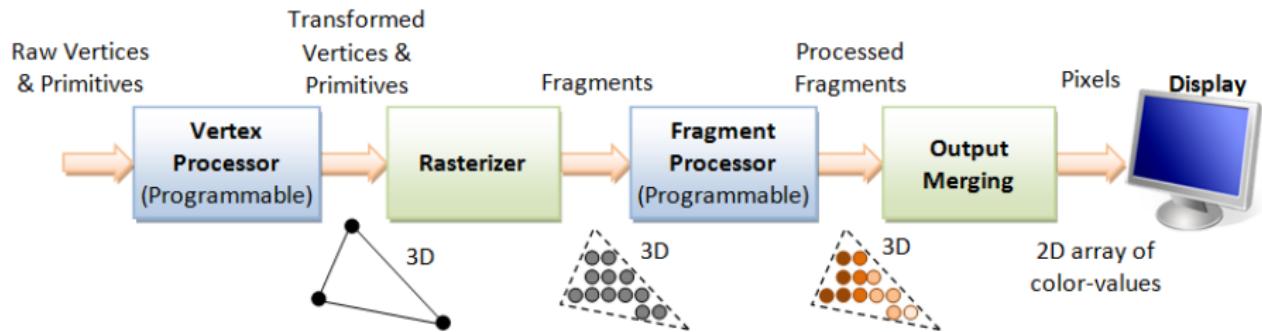
Tema 5: Física y colisiones. Efectos especiales.

- 5.1 Introducción a los motores físicos.
- 5.2 Interacción con dispositivos de entrada y dispositivos de salida.
- 5.3 Técnicas de optimización.
- 5.4 Personalización de fuerzas
- 5.5 Efectos especiales y técnicas volumétricas.
- 5.6 Shaders de vértices y técnicas avanzadas.

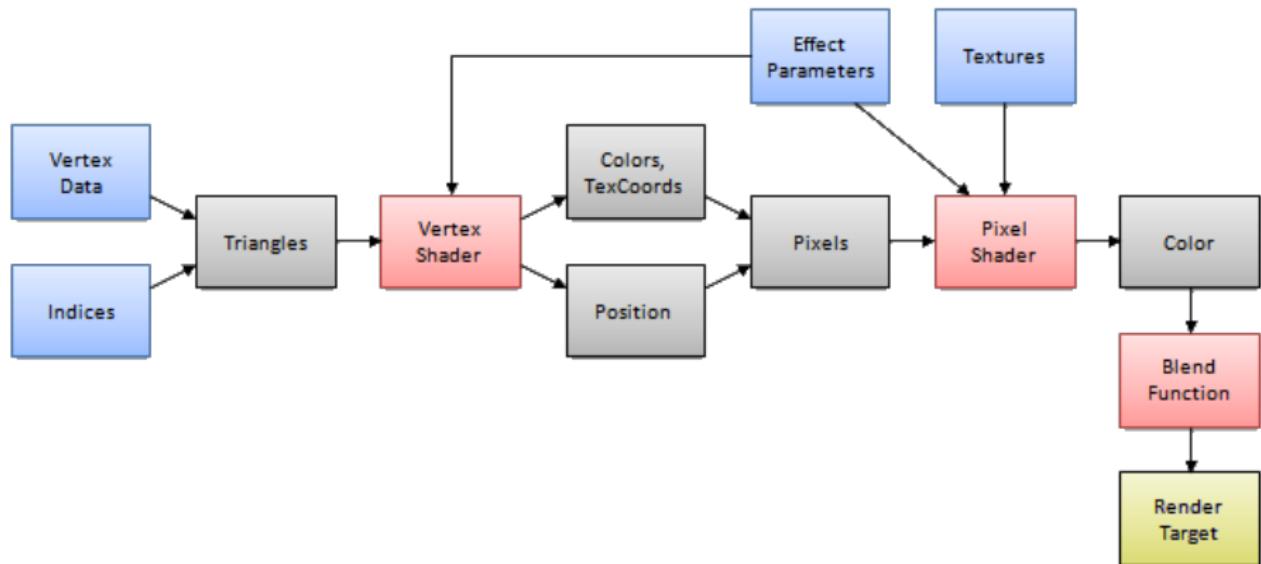
5.6 Shaders de vértices y técnicas avanzadas.



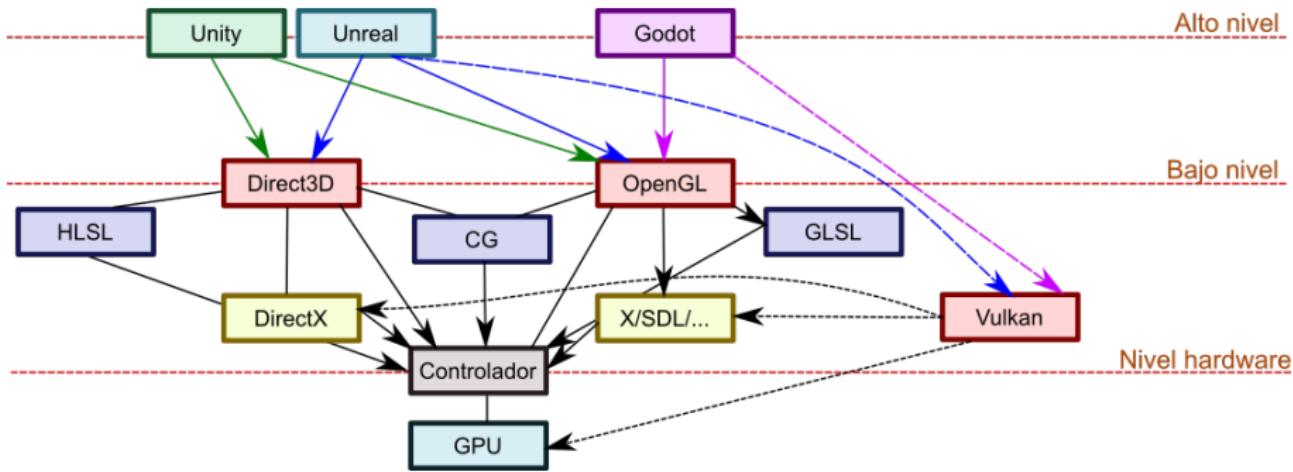
Esquema general de la GPU (I)



Esquema general de la GPU (II)



Alto y bajo nivel



Shaders a bajo nivel

```

22 SH_FracturedDepot(E_Gui
23 {
24     Value_Text;
25
26     // Load the shader
27     shader = new Shader("shaders\\fracturedDepot.frag");
28     shader->setUniform("tex0", 0);
29     shader->setUniform("tex1", 1);
30     shader->setUniform("tex2", 2);
31     shader->setUniform("tex3", 3);
32
33     active_shader = shader;
34 }
35
36 Gui->SetRect(100, 100, 300, 200);
37 Gui->SetRect(100, 200, 300, 300);
38 Gui->SetRect(100, 300, 300, 400);
39 Gui->SetRect(100, 400, 300, 500);
40 Gui->SetRect(100, 500, 300, 600);
41 Gui->SetRect(100, 600, 300, 700);
42 Gui->SetRect(100, 700, 300, 800);
43 Gui->SetRect(100, 800, 300, 900);

44 isChanged = IsChanged || isNormal == false;

45 fSize = new char[1000];
46 fSize[0] = '\0';
47 fSize[1] = '\0';

48 fValue = new char[1000];
49 fValue[0] = '\0';
50 fValue[1] = '\0';

51 fName = new char[1000];
52 fName[0] = '\0';
53 fName[1] = '\0';

54 fType = new char[1000];
55 fType[0] = '\0';
56 fType[1] = '\0';

```

code

```
File Edit View Menus Help Preferences Apache
New Open Save As... Guardar Guardar como... Importar Cerrar Diccionario
1 uniform sampler2D tex;
2 uniform sampler2D tex0;
3 uniform sampler3D texNormal;
4 uniform float ambientIntensity;
5 uniform vec3 upLightDir;
6 uniform vec3 upLightColor;
7 varying vec3 lightDir, halfVector;
8
9 void main()
10 {
11     vec3 trans, color, matColor, normal;
12     vec3 n, halfV;
13     float shadeR, shadeG;
14     vec4 sh;
15
16     // Out of bounds on gl_InvocationID
17     if (gl_TexCoord[0].x == 1.0) { gl_TexCoord[0].x = 0.0;
18     if (gl_TexCoord[0].y == 1.0) { gl_TexCoord[0].y = 0.0;
19     if (gl_TexCoord[0].z == 1.0) { gl_TexCoord[0].z = 0.0; }
20     }
21
22     if (color.y <= 0.0, 0.0, 0.0, 0.0);
23     discard;
24 }
```

frag

```
File Edit View Source Run Renderers Preferences Agenda
New Open Save Guard Guards Create Import Other Desktop
1 varying vec3 LightDir, halfVector;
2
3 void main()
4 {
5     // normal is a NormalMatrix * gl_Normal
6     gl_TexCoord[0] = gl_TextureCoord[0]; gl_MultiTexCoord0 // position for texels
7
8     LightDir = normalize(vec3(gl_LightSource[0].position));
9
10    // Normalize the halfVector to pass it to the fragment shader -f
11    halfVector = normalize(gl_LightSource[0].halfVector.xyz);
12
13    gl_Position = ftransform();
14
15 }
```

vert

Shaders a alto nivel

Distintos tipos:

- Material.
- Iluminación.
- Viewport/Canvas.
- Otros: partículas, propósito general, etc.

Ámbito de variables

Distinto ámbito:

- **Variables por vértices:** Variables que se pasan desde CPU al *vertex shader*.
- **uniform:** Variables que se pasan desde CPU.
- **varying:** Variables que se pasan desde el shader de vértices al de fragmentos (interpolación lineal).
- **Variables locales:** Variables que existen solamente en la función definida (típicas funciones estandar son *vertex*, *fragment* y *light*).
- **in:** variables de solo-lectura de los shaders.
- **out:** variables de solo-escritura de los shaders.
- **inout:** variables de lectura y escritura intrínsecas a los algoritmos de shaders.

¡No hay variables de «salida» a CPU!

Tipos de variables

Tipos comunes (y no tan comunes):

- **vec2,3,4:** vectores, vec3 y vec4 también sirven como colores. Se puede acceder a sus componentes: vector.x, vector.rg, vector.xyzw
- **mat4:** matrices 4x4 (transformaciones geométricas).
- **float,int,bool:** $1 \neq 1.0$
- **sampler1D,2D,3D:** un mapa o textura, se puede acceder a ella mediante la función *texture(sample2d, uv)*, devuelve un color de 4 dimensiones (rgba).

Lenguaje

Los lenguajes varían pero suelen imitar a C.

¡Cuidado! Los procesadores son poco potentes:

- **for, if, switch-case**, deberían ser evitados siempre que sea posible...

Procesadores vectoriales:

- Usar **vecX, matX, samplerXD**, siempre que se pueda...

Ejemplo: terreno procedural

- Una textura de ruido (Gris) nos indica la elevación.
- Una segunda textura (RGB) de ruido sirve para obtener las normales, o calcularla ?...
- Una tercera textura (RGB) nos da el color del terreno, o la elevación ?...

https://docs.godotengine.org/en/stable/tutorials/shading/shading_reference/shading_language.html

https://docs.godotengine.org/en/stable/tutorials/shading/shading_reference/spatial_shader.html

Obtener información de las texturas: múltiples pasadas

La única forma de capturar datos es leer la textura resultante (GPU → CPU).

Es típico encadenar pasadas:

- ① **[PRIMERA PASADA]** Preparar datos en CPU, transferir a GPU.
- ② Hacer algoritmo GPU y visualizar textura.
- ③ Capturar textura (viewport) y transferir a CPU (o no).
- ④ **[SEGUNDA PASADA]** Utilizar textura como entrada para el nuevo algoritmo.
- ⑤ ...

Sistemas de Partículas

- **Emisor:** desde donde sale la partícula. Puede haber sub-emisores.
- **Función de actualización:** donde se actualiza el estado/comportamiento de la partícula.
- **Vida de la partícula:** tiempo en el que la partícula es visible y tiene comportamiento.
- **Dibujado:** cada partícula puede ser cualquier objeto 3D con cualquier material.

<https://www.youtube.com/watch?v=4VDNBTF9mu0>

<https://www.youtube.com/watch?v=aNVviTECNM0>

Ejemplo de shaders y partículas

- GPU + partículas:
 - ▶ <https://godotengine.org/article/improvements-gpuparticles-godot-40>
 - ▶ https://docs.godotengine.org/es/stable/classes/class_particles.html
- Mapas de elevación y colisiones:
 - ▶ <https://godotengine.org/storage/app/media/4.0/particles/window.mp4>
- Mapas de elevación y hierba:
 - ▶ <https://www.youtube.com/watch?v=uMB3-g8v1B0>
 - ▶ <https://github.com/BastiaanOlij/godot-grass-tutorial>