(algunas) Aplicaciones de las Matemáticas en Informática Gráfica

Carlos Ureña Almagro

Dpt. Lenguajes y Sistemas Informáticos
ETSI Informática y de Telecomunicación
Universidad de Granada

2 de marzo de 2011

Proyecto de Innovación Docente
ORIENTAMAT

Estructura de la presentación

(algunas) Aplicaciones de las Matemáticas en Informática Gráfica

---

1. Informática gráfica y *rendering* realista
2. Aplicaciones y ejemplos de *rendering* realista
3. El modelo digital del escenario y los objetos
4. Visualización usando *Z-buffer*
5. La técnica de *ray-tracing*
6. Cálculo de iluminación global
7. IG en la Universidad de Granada

Sección 1

Informática gráfica y *rendering* realista

Definición de **Informática Gráfica**

La **Informática Gráfica** (IG) es una especialidad dentro de la Informática dedicada a estudiar las técnicas y metodologías adecuadas para el desarrollo de software relacionado con la creación, adquisición, manipulación y presentación de información de carácter principalmente visual.

La IG usa conceptos de otros áreas de la Informática: Ingeniería del Software, Programación, Técnicas de Interacción Hombre-Ordenador, y también de otras disciplinas: Psicología, Biología, Física y, por supuesto **Matemáticas**.

Procesos en los que se usa este software

Principalmente se trata del diseño de software útil para:

- La creación o diseño de modelos digitales de objetos reales,
- La adquisición de dichos modelos a partir de los objetos existentes
- La manipulación de los modelos adquiridos o diseñados
- La producción de imágenes y vídeos, completa o parcialmente sintéticos, a partir de los modelos.
- La visualización o manipulación interactiva de dichos modelos.

Computación Visual y el CAM

La Informática Gráfica se encuadra en un ámbito de conocimiento más amplio que puede llamarse **Computación Visual** (**Visual Computing**, y que incluye, entre otras disciplinas

CAD= **Computer Aided Design** (diseño asistido por ordenador de nuevos objetos)

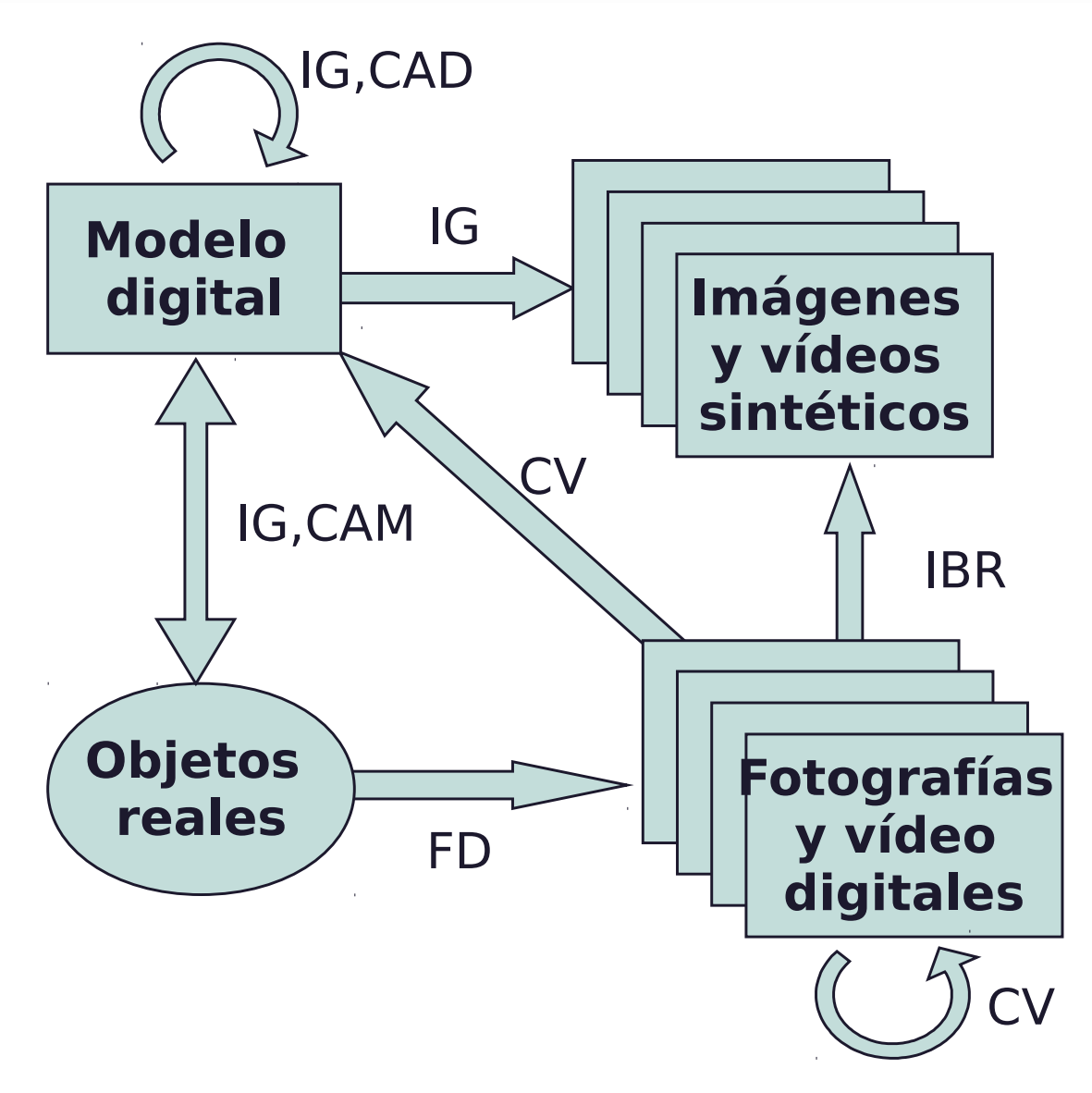
CAM= **Computer Aided Manufacturing** (fabricación de objetos asistida por ordenador)

CV= **Computer Vision** (obtención de modelos a partir de fotografías o vídeos)

IBR= **Image Based Rendering** (producción de imágenes nuevas a partir de imágenes existentes, sin obtener modelos completos)

FD= Fotografía y vídeo Digital (obtención directa de imágenes)

Campos relacionados con o incluidos en IG



IG en la Universidad de Granada

El Grupo de Investigación en Informática Gráfica trabaja, entre otros, en estos campos:

- Rendering: producción de imágenes realistas a partir de modelos
- Non-photorealistic rendering : producción de imágenes no realistas a partir de modelos o fotografías
- Adquisición de modelos: obtención de modelos mediante escáner láser
- Realidad Virtual: sistemas interactivos e inmersivos de visualización de modelos o entornos

Creación de imágenes similares a fotografías

Un objetivo maximalista, pero no alcanzable hoy en día es:

> *A partir de un modelo digital de un entorno real, producir en el observador una experiencia visual similar a la que tendría estando situado en dicho entorno real.*

Un objetivo menos ambicioso, pero útil y realizable, es:

> *A partir de un modelo digital de un entorno real, producir una imagen por ordenador que sea indistinguible de una fotografía de dicho entorno real (Shirley,1990)*

Otros objetivos de la sintesis realista

El realismo visual no siempre es el objetivo principal. Por ejemplo:

- En los videojuegos, se busca el máximo de realismo dentro de los limites impuestos por el tiempo de calculo por cuadro (1/25 de segundo)
- En visualización científica o técnica, se busca mostrar con la mayor claridad posible determinadas características del modelo digital, que no se verían en una imagen realista.
- En algunos videojuegos, películas de animación, o productos artísticos, el realismo esta supeditado a la expresividad, a la intencionalidad artística, o a la imitación de técnicas clasicas de animación (rendering no fotorealista, cartoon rendering)

Imágenes real/sintética de un entorno complejo

La complejidad en tiempo del cálculo depende de la complejidad del escenario, aquí vemos un ejemplo de un escenario complejo.

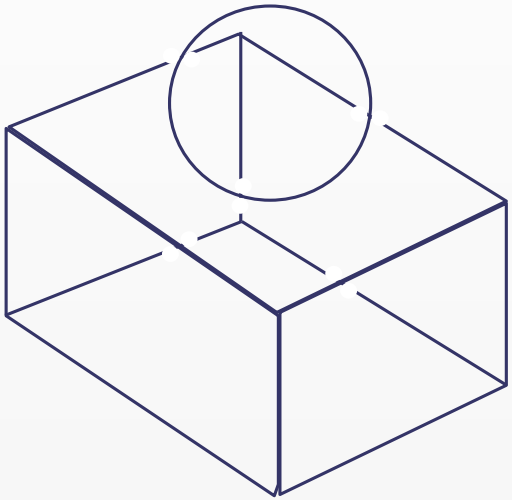
fotografía

imagen generada

☞ http://www.mpi-inf.mpg.de/resources/atrium/atrium_old/welcome.html

El proceso de rendering

Nuestro objetivo final es el diseño de algoritmos para calcular imágenes digitales a partir de modelos de objetos almacenados en memoria.



modelo geométrico
+ colores, luces

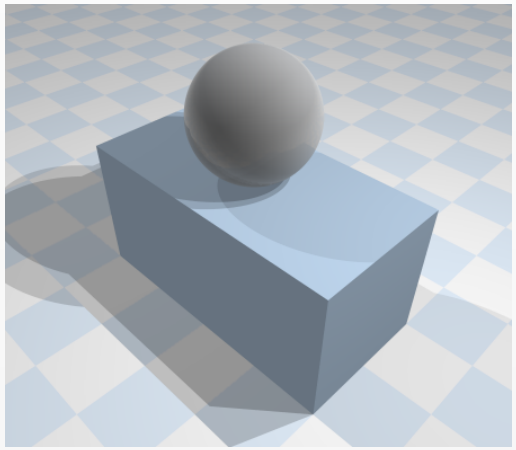


imagen generada

Sección 2

Aplicaciones y ejemplos de *rendering* realista

Aplicaciones del rendering realista (1/2)

- Videojuegos
- Arquitectura
- Decoración
- Marketing y publicidad
- Iluminación de exteriores e interiores
- Diseño de luminarias
- Diseño en general
- Diseño industrial

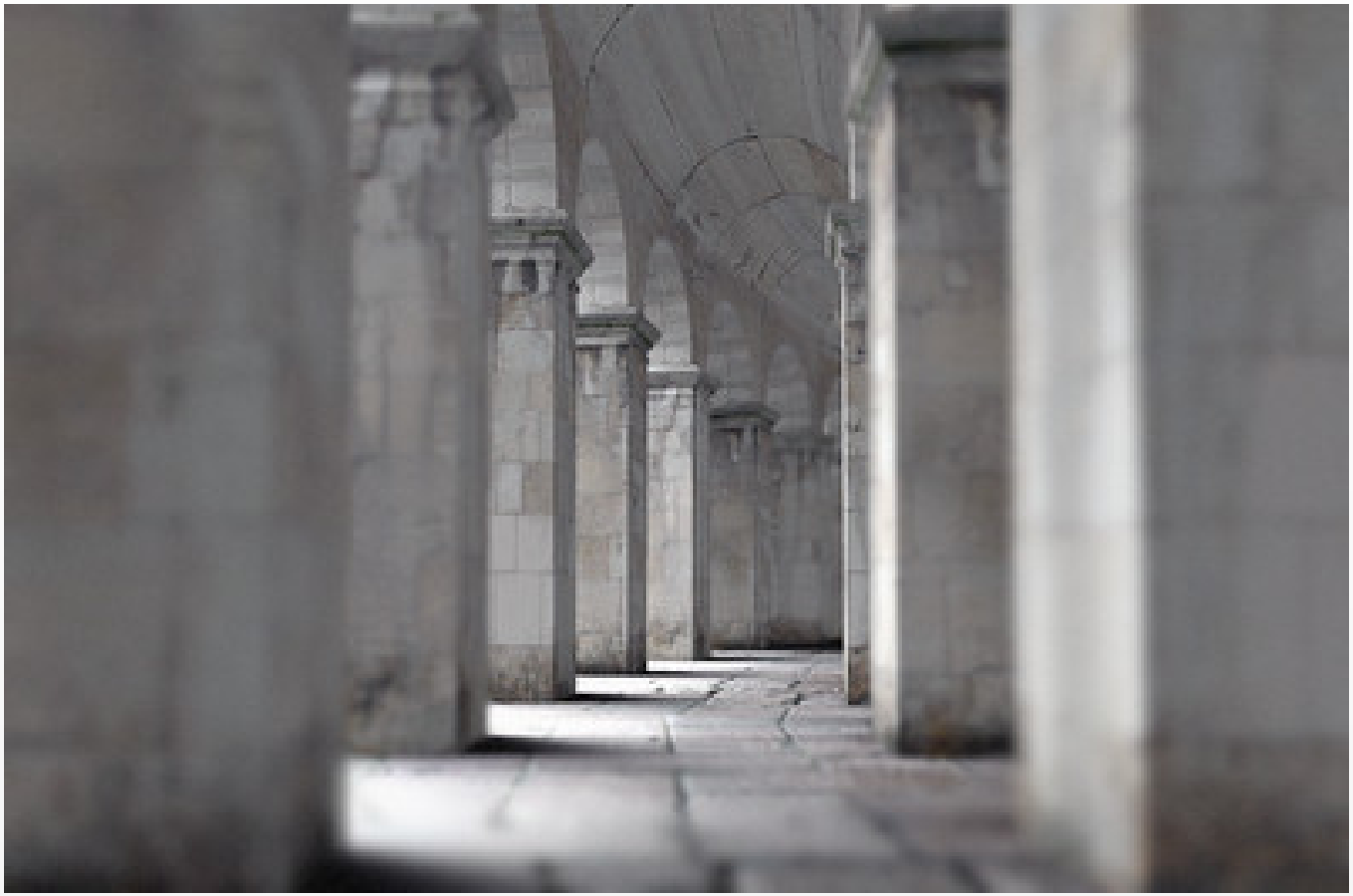
Aplicaciones del rendering realista (2/2)

- Cine completamente generado por ordenador
- Cine con elementos generados por ordenador
- Cine de animación clásico generado por ordenador
- Decorados virtuales para televisión
- Ingeniería civil
- Urbanismo
- Visualización Científica

Arquitectura

Urbanismo

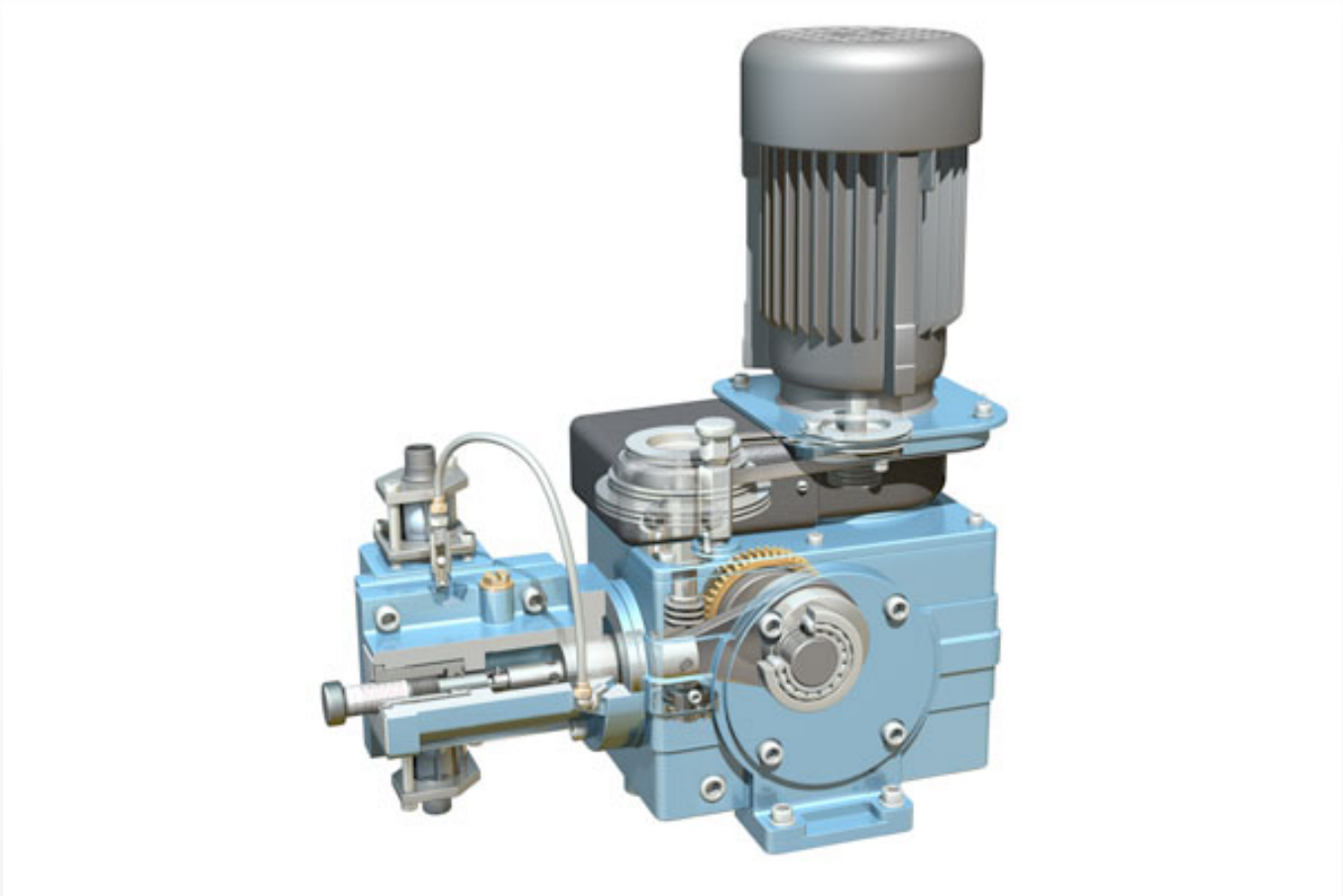
Patrimonio histórico



Diseño de interiores

Diseño de automóviles

Diseño industrial



Videojuegos

Videojuegos

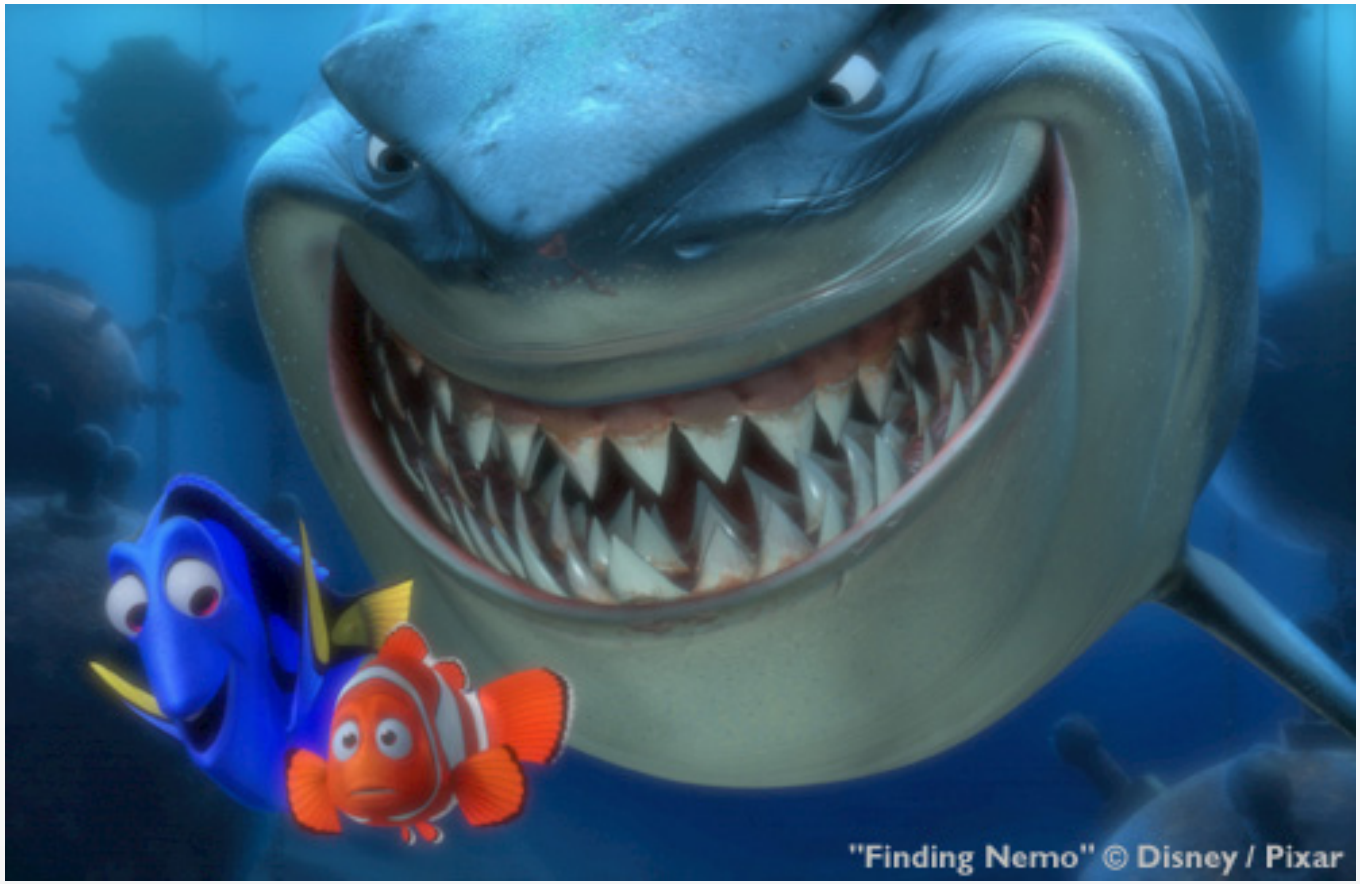
Efectos especiales en cine convencional

Inspector Gadget © 1999 Walt Disney Pictures.
Visual Effects by Dream Quest Images.

Cine generado por ordenador

Star Wars: Episode II, Attack of the Clones © 2002 Lucasfilm Ltd & TM. All rights reserved.
Photo Credit: Industrial Light & Magic.

Cine de animación por ordenador



Sección 3

El modelo digital del escenario y los objetos

Representación con modelos de caras planas

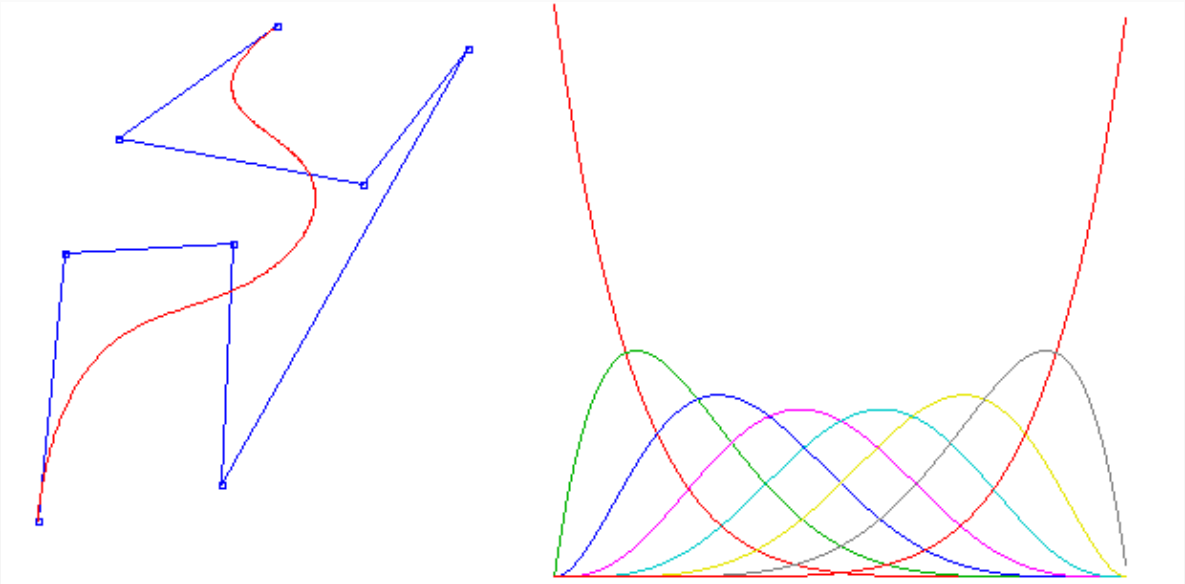
Es la forma más universal de representar objetos

- Un objeto es un volumen delimitado por un conjunto de polígonos planos (triangulos)
- Los vértices y las aristas forman un grafo
- Se usan estructuras de datos de diversos tipos para representar grafos (la forma más simple es mediante tablas de vértices, aristas y caras)

Los resultados de la **teoría de grafos** se usan para esto. Por ejemplo: un algoritmo para calcular cuantas componentes conexas hay.

Curvas de Beziers

Una forma de aproximar superficies curvas mediante caras planas es usar curvas obtenidas como una combinación lineal de funciones base, cada una de las cuales es un **spline** (función polinomial a trozos). Este un ejemplo de curva de **Beziers** (usando un vector de puntos de control):



demostración interactiva ☞ http://ibiblio.org/e-notes/Splines/Bezier.htm

Curvas de Beziers

Un punto $\mathbf{p}$ es una función de $t \in [0,1]$, y se expresa como una combinación lineal de determinadas funciones base, siendo los pesos los $n+1$ puntos de control $\mathbf{c}_i$:

$$\mathbf{p}(t) \;=\; \sum_{i=0}^{n} B_i^n(t)\,\mathbf{c}_i$$

donde las funciones base se pueden definir recurrentemente como:

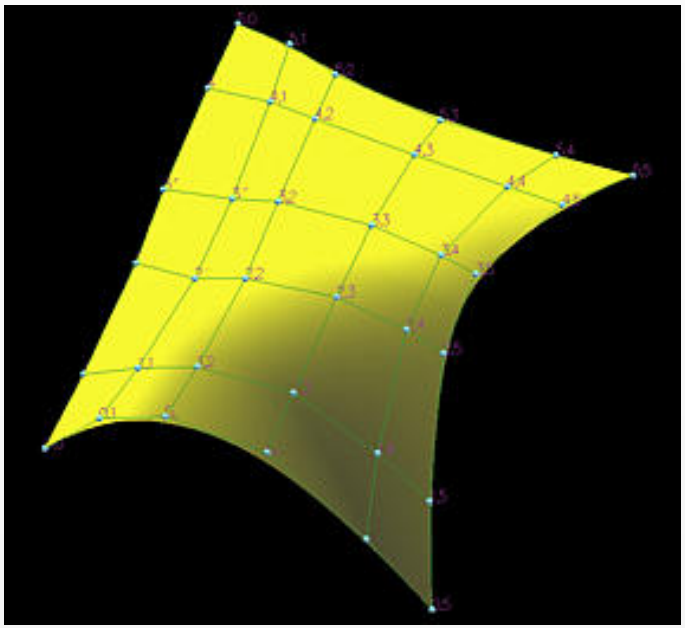
$$B_i^j(t) \;=\; \begin{cases} 1 & \text{si } i=0 \text{ y } j=0 \\ (1-t)B_i^{j-1}(t) \;+\; tB_{i-1}^{j-1}(t) & \text{si } 0 \le i \le j \\ 0 & \text{en otro caso} \end{cases}$$

también se pueden obtener directamente como los **polinomios de Bernstein**

$$B_i^n(t) \;=\; \frac{n!}{i!(n-i)!}\,t^i(1-t)^{n-i}$$

Superficies B-Spline

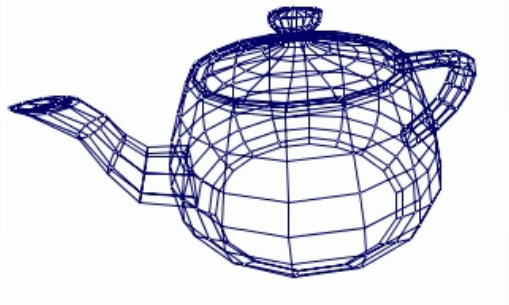
A partir de la composición de dos curvas uni-dimensionales, se obtienen supericies bidimensionales (usando una malla de puntos de control). Un ejemplo son las **superficies B-Spline** (generalizaciones de curvas de Beziers):

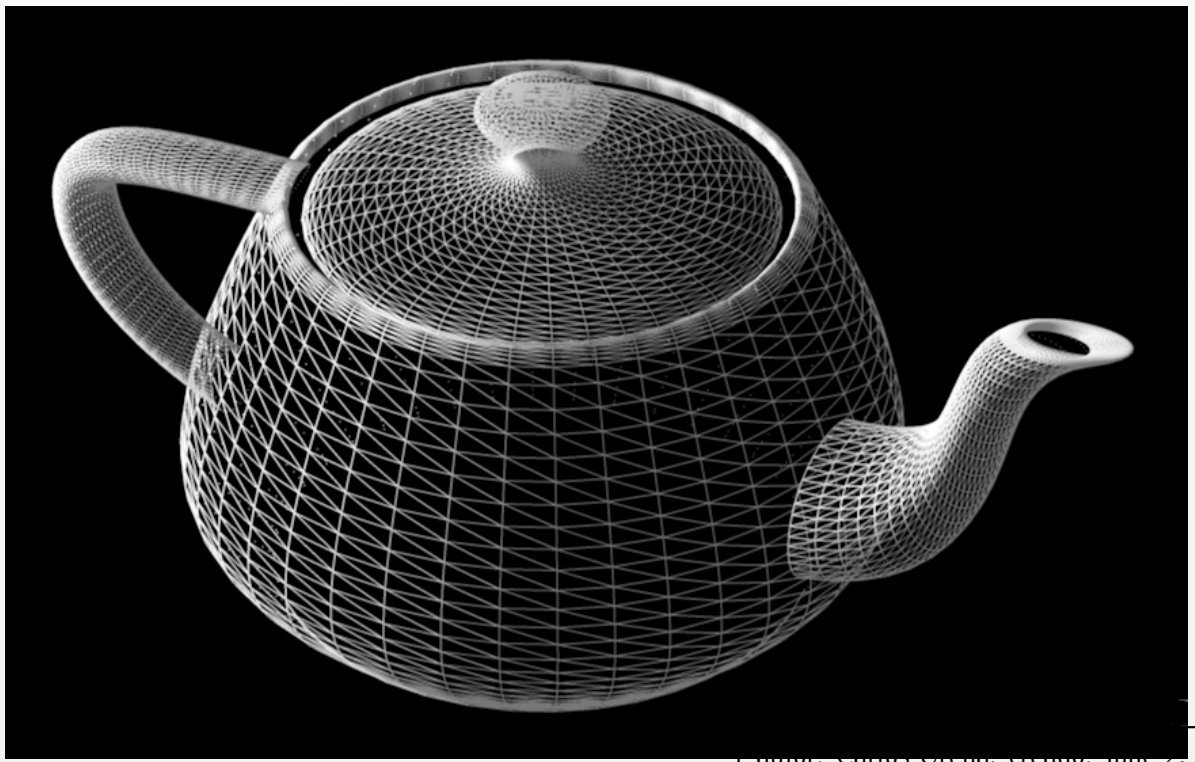


☞ http://www.cs.mtu.edu/ shene/COURSES/cs3621/NOTES/surface/bspline-construct.html

Curvas y Superficies de Beziers, B-Spline y NURBS

Un ejemplo de uso de superficies B-Spline:





Propiedades de las curvas

En estas familias de curvas y superficies se pueden diseñar curvas para las cuales es posible especificar:

- los puntos de control por donde pasa la curva o superficie
- la dirección del vector tangente en cada punto (o la orientación de la superficie)
- la curvatura de la curva o superficie en cada punto

Existen otras familias de construcciones, como las **superficies de subdivisión de Catmull-Clark**.

Modelos basados en la ecuación implicita

Un objeto puede estar definida por una **ecuación implícita**, es decir, existirá una función real F tal que para cualquier punto $\mathbf{p}$ se cumple:

$$F(\mathbf{p}) = \begin{cases} < 0 & \text{si } \mathbf{p} \text{ está dentro del objeto} \\ 0 & \text{si } \mathbf{p} \text{ está en la superficie del objeto} \\ > 0 & \text{si } \mathbf{p} \text{ está fuera del objeto} \end{cases}$$

esta es una forma bastante general de definir un objeto. Puede usarse, p.ej., para modelar fractales o para *isosuperficies de campos escalares*, como los campos basados en un potencial:

$$F(\mathbf{p}) = \frac{1}{d^2} - \sum_{i=1}^{n} \frac{1}{\|\mathbf{p} - \mathbf{c}_i\|^2}$$

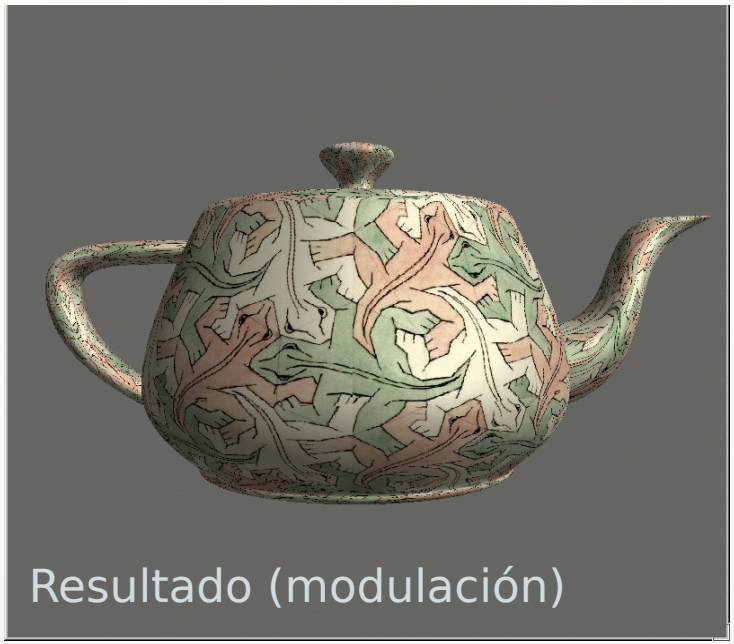
(modela una isosuperficie a una distancia (aprox.) d de un conjunto de puntos $\mathbf{c}_i$)

Modelos con texturas

El grado de realismo de un modelo se puede mejorar usando texturas:

textura sin sombreado

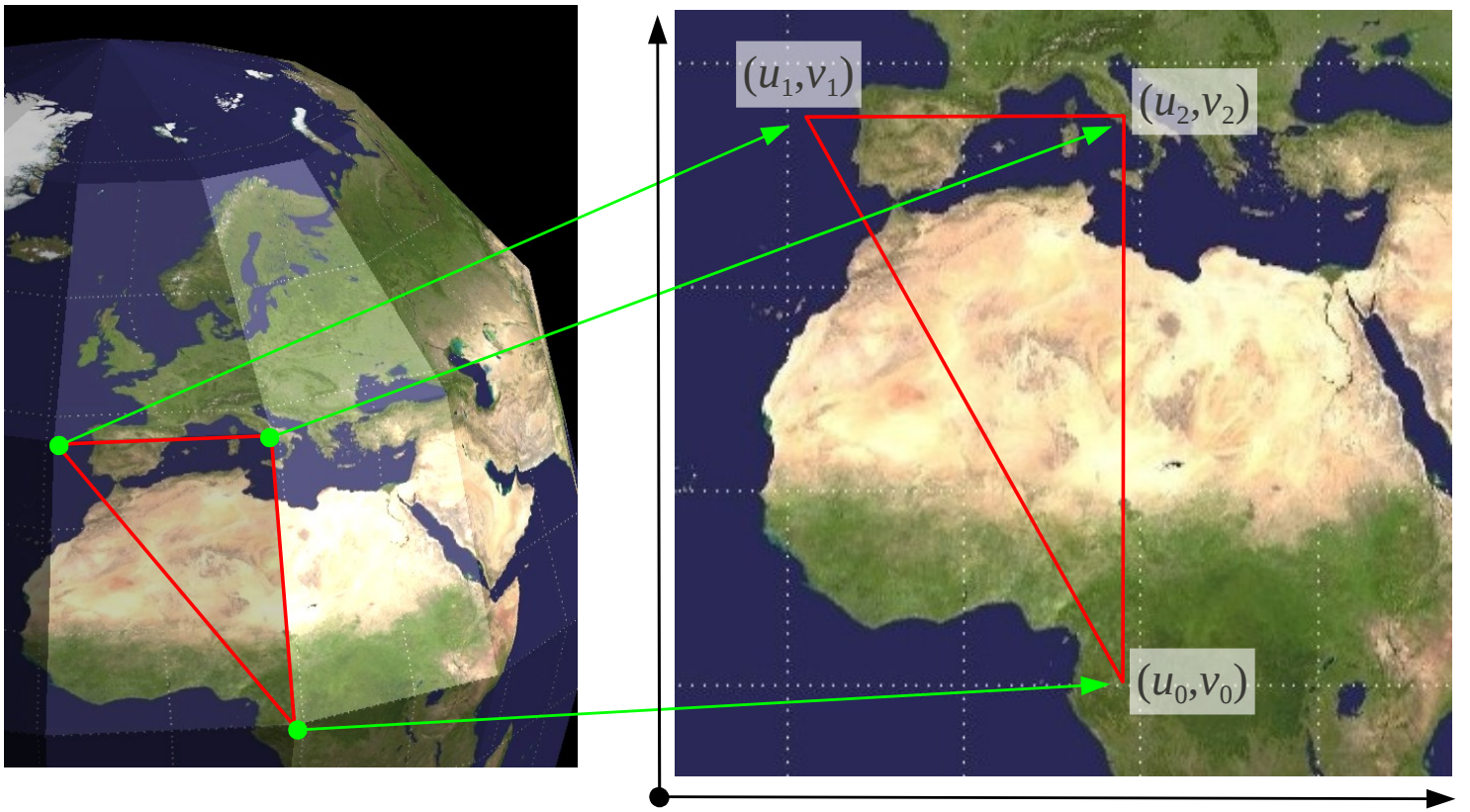
sombreado sin textura



se usa una imagen ("textura") que se "pega" a la superficie

Asignación de coordenadas de texturas

Para hacer corresponder la textura con el objeto se deben asignar coordenadas de textura (pares de la forma $(u, v) \in [0, 1]^2$) en los vértices:



hay un mapa M que asigna puntos a coords. de textura $M(u, v) = p$ (y

Aplicaciones del *flujo de Ricci* a texturas

- Existen diversas formas de asignar coordenadas de textura, el problema es que no hay métodos automáticos que garantizen que la textura no se deforme.
- Para que la textura no aparezca deformada, se puede intentar que el mapa de texturas sea **holomorfo** (es decir, derivable en sentido complejo), lo cual equivale a que el mapa de textura es **conforme** (conserva los ángulos)
- Recientemente se han aplicado el concepto de **flujo de Ricci** para diseñar un proceso similar en mallas de polñigonos, llamado **flujo de Ricci discreto**, que sirve a estos fines.
- el flujo de Ricci es un concepto esencial en la demostración de la **conjetura de Poincaré** realizada por Gregori Perelman.

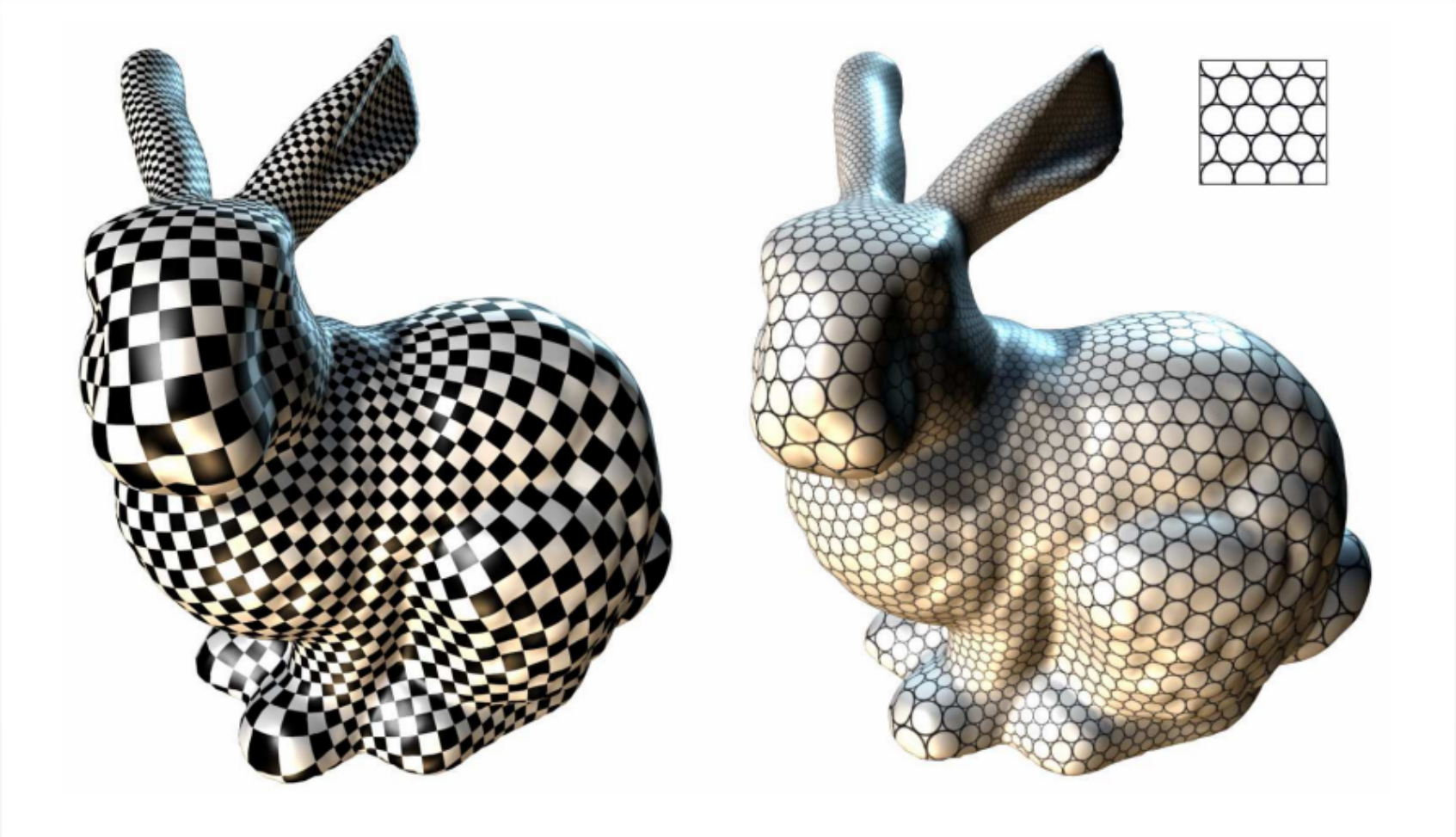
Aplicaciones del *flujo de Ricci* a texturas

esta técnica permite encontrar parametrizaciones conformes de cualquier superficie topológicamente equivalente a una esfera:

☞ http://www.cs.sunysb.edu/ vislab/papers/RicciFlow.pdf

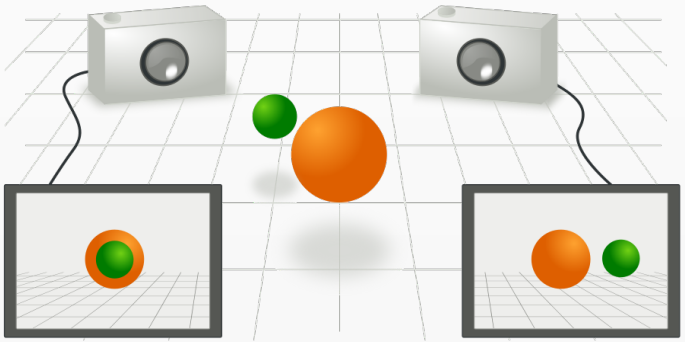
(imagen obtenida del artículo de Miao Jin y otros *Discrete Surface Ricci Flow: Theory and Applications* (2007))

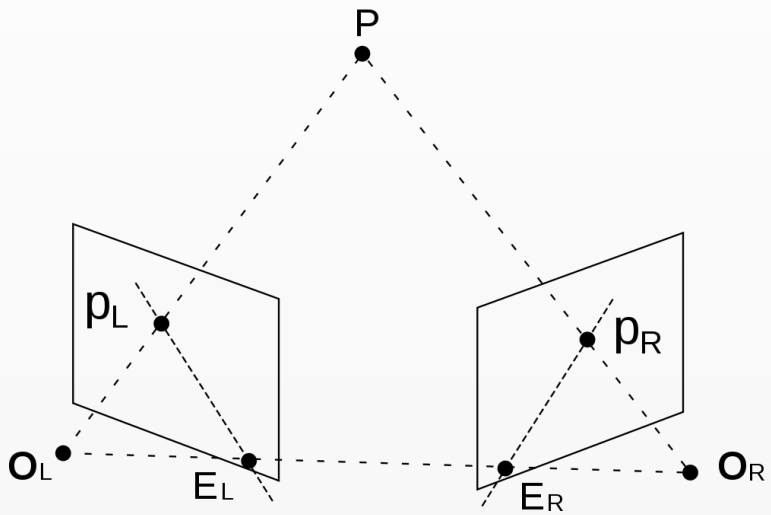
Aplicaciones del *flujo de Ricci* a texturas



Obtención de modelos por fotografía digital

Se pueden obtener a partir de pares de fotografías digitales





☞ http://en.wikipedia.org/wiki/Epipolar_geometry

Se usan los resultados de la **geometría epipolar**. Es necesario identificar un conjunto de puntos en las dos imágenes, y esta correspondencia permite obtener su posición (incluso sin que las cámaras estén calibradas).

Obtención de modelos por escáner láser

También se pueden obtener modelos usando un escáner láser, que mide las distancias desde un foco a un conjunto de puntos en una superficie

adquisición de modelos 3D del patio de los leones de la Alhambra (realizado por Pedro Cano y otros)

☞ http://www.alhambra-patronato.es/fileadmin/pdf/Taller_CICOP_FINAL_V3.pdf

Obtención de modelos por escáner láser

La obtención de los modelos requiere realizar distintas tomas del escáner (por ejemplo, en los leones, una a cada lado)

- Para cada toma se obtiene una nube de puntos
- Para cada nube se debe obtener una malla (o parche) de polígonos (p.ej.: mediante variantes de la **triangulación de Delauny**).
- Se debe encontrar como hacer coincidir estos "parches", mediante transformaciones rígidas para que encajen entre sí en las partes comunes
- Se usan **algoritmos de minimización** que tratan que encontrar la transformación que hace menor una determinada **métrica de distancia** entre las nubes de puntos.

Transformaciones y coordenadas homogéneas

En visualización y animación, es frecuente la necesidad de especificar transformaciones sobre objetos tridimensionales (típicamente sobre mallas de polígonos)

- Las posiciones de los vértices (en 3D) se almacenan usando **coordenadas homogéneas**
- Un punto 3D se representa mediante una tupla de 4 valores reales (el último distinto de cero). El punto de coordenadas homogéneas (x, y, z, w) se corresponde con el punto de coordenadas cartesianas $(x/w, y/w, z/w)$.
- Las transformaciones lineales en 3D (**rotación**, **escalado** y **traslación**), se pueden representar como **matrices 4x4** en coordenadas homogéneas (la traslación, por ejemplo, no es lineal en coordenadas cartesianas).
- La composición de transformaciones se corresponde con la composición de matrices

Ejemplo: Rotación seguida de traslación

Supongamos una rotación R (α radianes en torno al eje Z) seguida de una traslación D por (t_x, t_y, t_z) Un punto $\mathbf{p} = (x, y, z, w)$ se transforma en otro $\mathbf{q} = (x', y', z', w')$ mediante estas expresiones:

$$\begin{aligned} x' &= t_x + x\cos(\alpha) - y\sin(\alpha) \\ y' &= t_y + x\sin(\alpha) + y\cos(\alpha) \\ z' &= t_z \\ w' &= w \end{aligned}$$

matricialmente se puede escribir $\mathbf{q}^T = D\,R\,\mathbf{p}^T$, con más detalle:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotación usando quaterniones

En animación y en sistemas interactivos, a veces es necesario aplicar una sucesión de rotaciones dependientes del tiempo que llevan a un objeto desde una orientación determinada hasta otra. Si se emplea composición de rotaciones entorno a los ejes, se pueden dar singularidades que provocan errores numéricos.

Para solucionar este problema se pueden representar las rotaciones mediante el uso de **Quaterniones** (**Quaternions**). Estos objetos son una extensión de los números complejos (de 2 dimensiones) a 4 dimensiones. Un quaternion tiene la forma:

$$a + b\mathbf{i} + c\mathbf{j} + c\mathbf{k}$$

donde $\mathbf{i}$,$\mathbf{j}$ y $\mathbf{k}$ son quaterniones que cumplen:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1 \qquad \mathbf{ij} = \mathbf{k} \qquad \mathbf{jk} = \mathbf{i} \qquad \mathbf{ki} = \mathbf{j}$$

Rotación usando quaterniones

Al multiplicar un número complejo por otro (de módulo unidad), estamos aplicando una rotación al primero en el plano de Argand. Esto puede generalizarse al espacio tridimensional y a los quaterniones.

Un punto (x, y, z) se representa por el quaternión $\mathbf{p}$

$$\mathbf{p} = 0 + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

Una rotación un ángulo α alrededor de un eje $\mathbf{v} = (v_x, v_y, v_z)$ (con $\|\mathbf{v}\| = 1$) se representa por el quaternión $\mathbf{q}$ (unitario):

$$\mathbf{q} = \cos(\alpha/2) + \sin(\alpha/2)\left[v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}\right]$$

entonces el punto rotado (x', y', z') se representa por el quaternion $\mathbf{p}'$ dado por

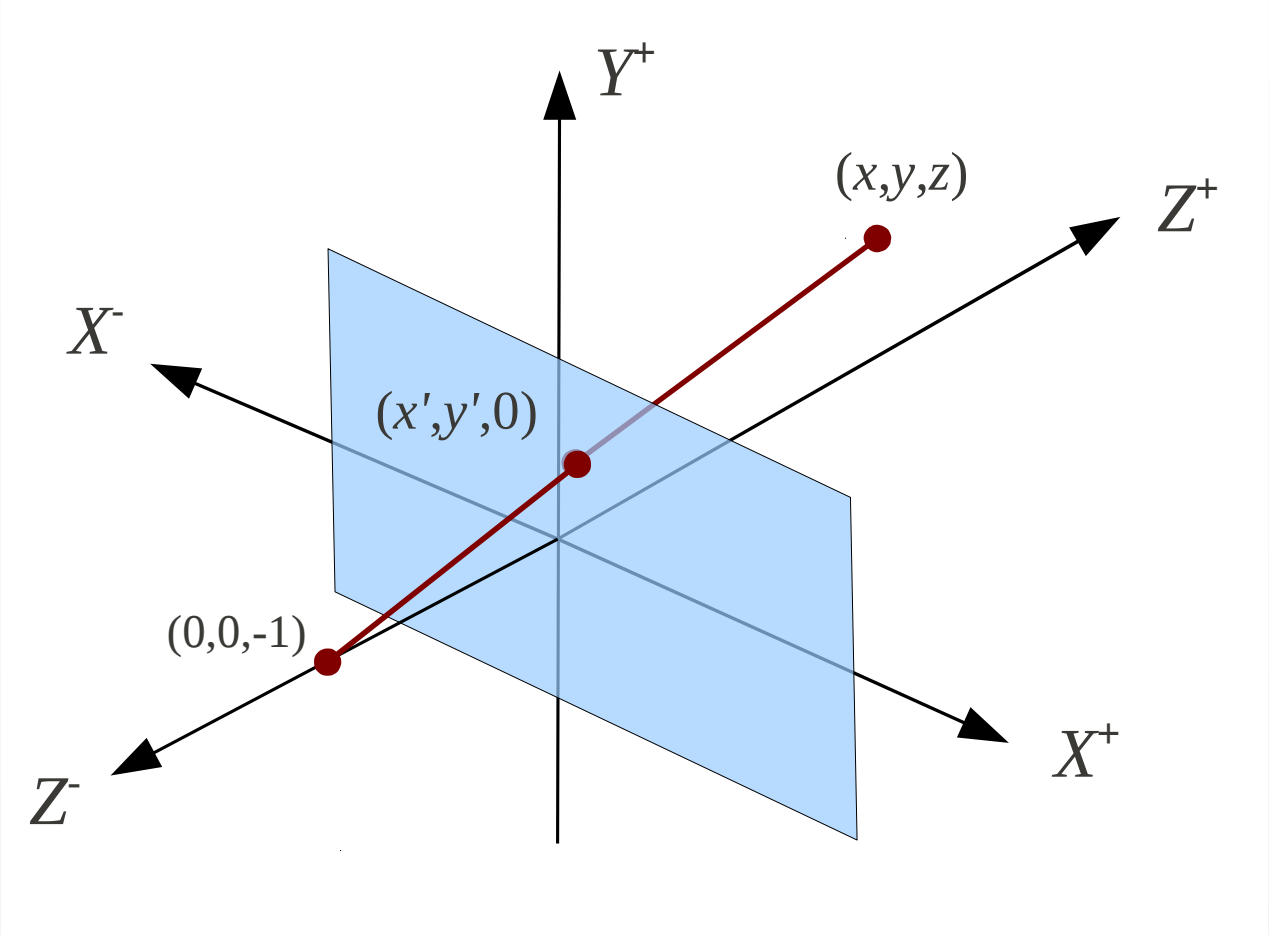
$$\mathbf{p}' = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^{-1}$$

Sección 4

Visualización usando *Z-buffer*

Visualización: la transformación de proyección

Para la visualización, es necesario **proyectar** puntos. Se usa la **geometría proyectiva**



La transformación de proyección en coords. euclídeas

Si suponemos que el foco de la proyección es el punto $(0,0,-1)$ y el plano donde se proyecta es el plano $z=0$, esto puede hacerse usando una transformación perspectiva como esta:

$$x' = \frac{x}{z+1} \qquad y' = \frac{y}{z+1} \qquad z' = 0$$

- Esta transformación no es lineal, no puede expresarse como una matriz
- La información sobre la profundidad se pierde (la coord. Z se pone a cero)

La transformación de proyección en coords. homogéneas

Para conservar un valor z' que nos indique la profundidad, podemos hacer $z' = z$. Además la proyección se puede expresar como una matriz 4x4 lineal, en coordenadas homogéneas:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

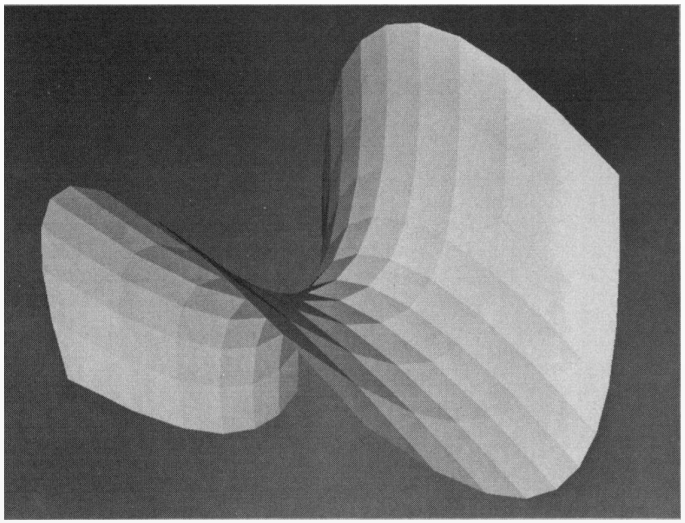
nótese que la división por $z+1$ se hace al recuperar las coordenadas cartesianas (x'', y'', z''):

$$x'' = \frac{x'}{w'} = \frac{x}{z+1} \qquad y'' = \frac{y}{z+1} \qquad z'' = \frac{z}{z+1}$$

(suponiendo $w = 1$)

Visualización: el algoritmo de Z-buffer

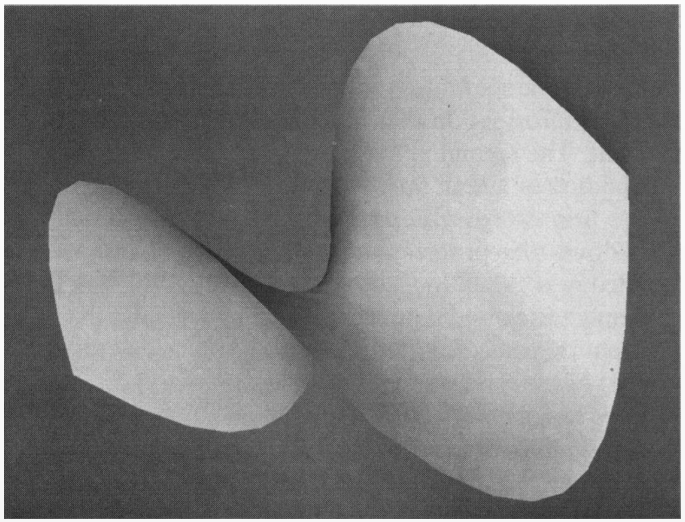
Dado un modelo formado por una malla de polígonos (típicamente triangulos), es necesario proyectar los vértices y obtener así los pixels en los que se proyecta cada triángulo.



Para determinar en que pixels se ve cada triángulo y en cuales está oculto, se puede usar una técnica conocida como Z-buffer.

Visualización: interpolación de Z y las intensidades

Para conocer la profundidad en cada pixel se puede hacer una interpolación de las profundidades en Z. También se pueden interpolar los colores de los pixels calculados en los vértices:



Todo esto se puede hacer de forma matemáticamente correcta y computacionalmente eficiente usando **interpolación hiperbólica**, (una interpolación lineal del valor $1/w$)

Visualización: coordenadas de texturas

- Para conocer el color de cada pixel es necesario interpolar en cada triangulo las coordenadas de textura.
- Esto se puede hacer usando interpolación hiperbólica, al igual que se hace para las intensidades y las profundidades en Z
- Estas interpolaciones, junto con el álgebra de matrices 4x4, se implementa en hardware en las tarjetas gráficas de los ordenadores personales y las consolas de videojuegos (los calculos se hacen muy eficientemente en hardware)
- Esto permite diseñar videojuegos interactivos realistas, en los cuales el proceso de calcula el color de un pixel se realiza aproximadamente 25 millones de veces por segundo (y no es el único cálculo que hay que hacer !)

Sección 5

La técnica de *ray-tracing*

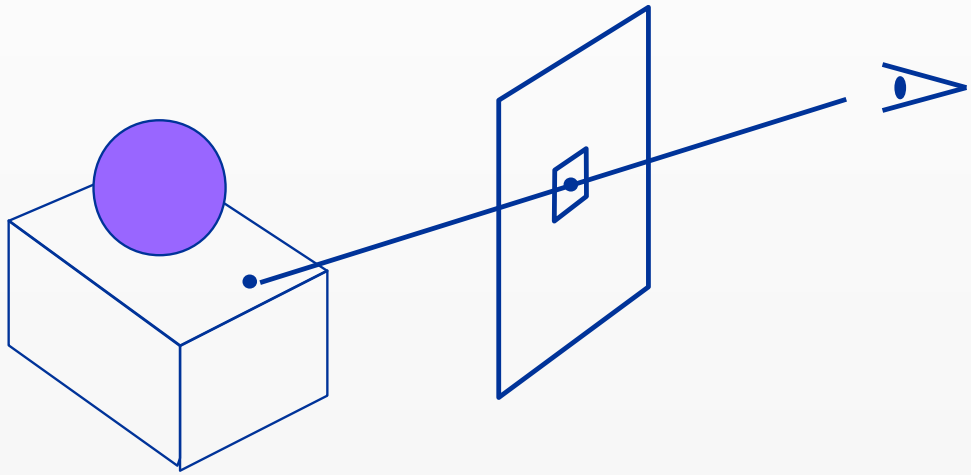
Ray-tracing

La técnica de Ray-tracing, ideada a principios de los 80, necesita más tiempo de cálculo pero produce mejores resultados

reproduce mejor los reflejos y las sombras arrojadas

Ray-tracing

Para cada pixel, es necesario encontrar la primera intersección de un semirecta con algún objeto de la escena



en cada pixel, hay que calcular las intersecciones con todos los objetos que forman la escena, y seleccionar la más cercana, si hay alguna. De esta forma se conoce que objeto se proyecta en el pixel.

Ray-Tracing: modelos basados en eq. implicita

Por otro lado en cada pixel es necesario encontrar la intersección de un semirecta de la forma

$$\mathbf{q} = \mathbf{o} + t\mathbf{v}$$

con cada objeto de la escena. Para ello se deben buscar los ceros de la función g

$$g(t) = F(\mathbf{o} + t\mathbf{v})$$

en algunos casos sencillos (p.ej., para las **cuádricas**) hay una solución analítica fácil. Si solo podemos evaluar g (y su derivada), podemos usar el **método de Newton**.

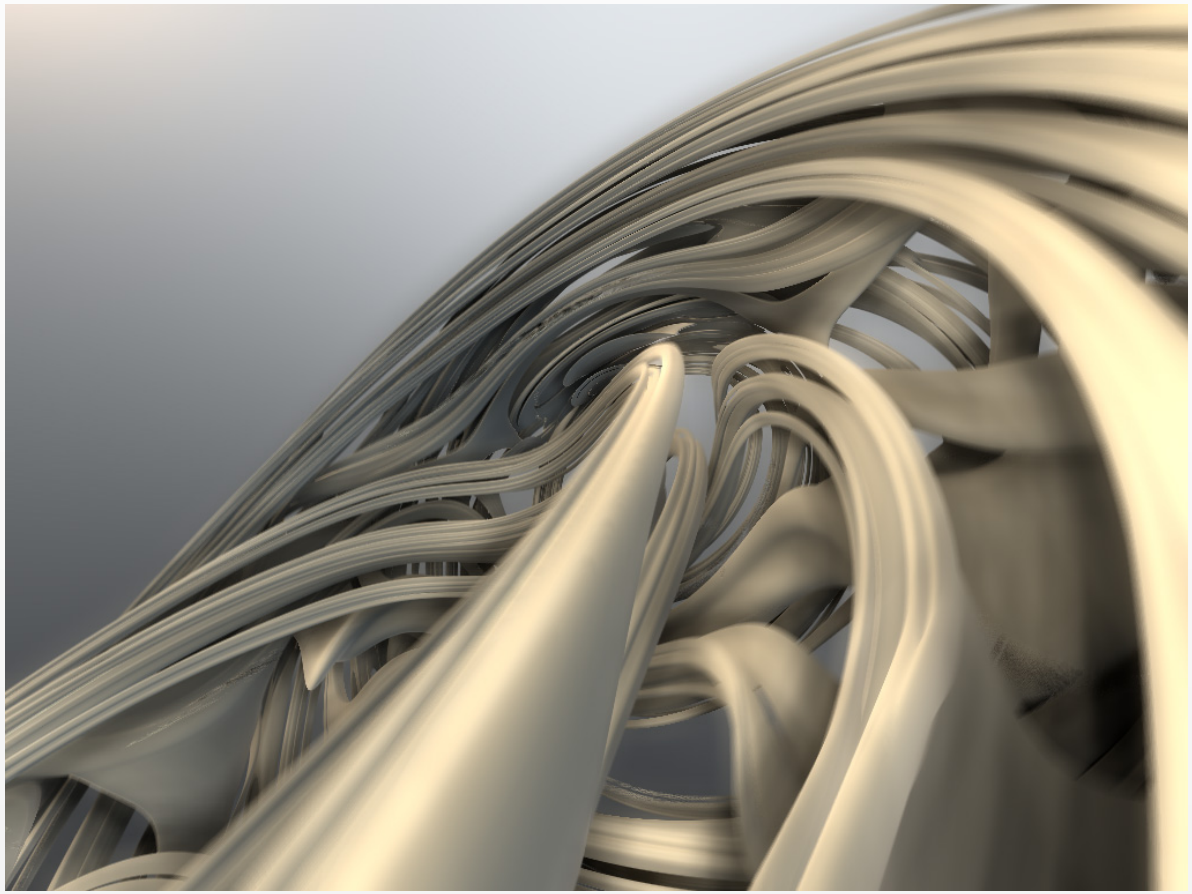
Ray-Tracing: ejemplo de isosuperficie

Esto puede usarse para visualizar una isosuperficie de un **campo escalar**, por ejemplo un campo escalar basado en una función potencial:

☞ http://www.codermind.com/articles/Raytracer-in-C++-Depth-of-field-Fresnel-blobs.html

Ray-Tracing: ejemplo de fractal en 3D

Un fractal (conjunto de Julia) es un tipo de objeto que únicamente puede ser modelado con su ecuación implícita



☞ http://www.iquilezles.org/www/articles/juliasets3d/juliasets3d.htm

Ray-tracing: tiempo de exposición

La técnica se puede extender para simular el tiempo de apertura finita de las cámaras:

Ray-tracing: profundidad de campo

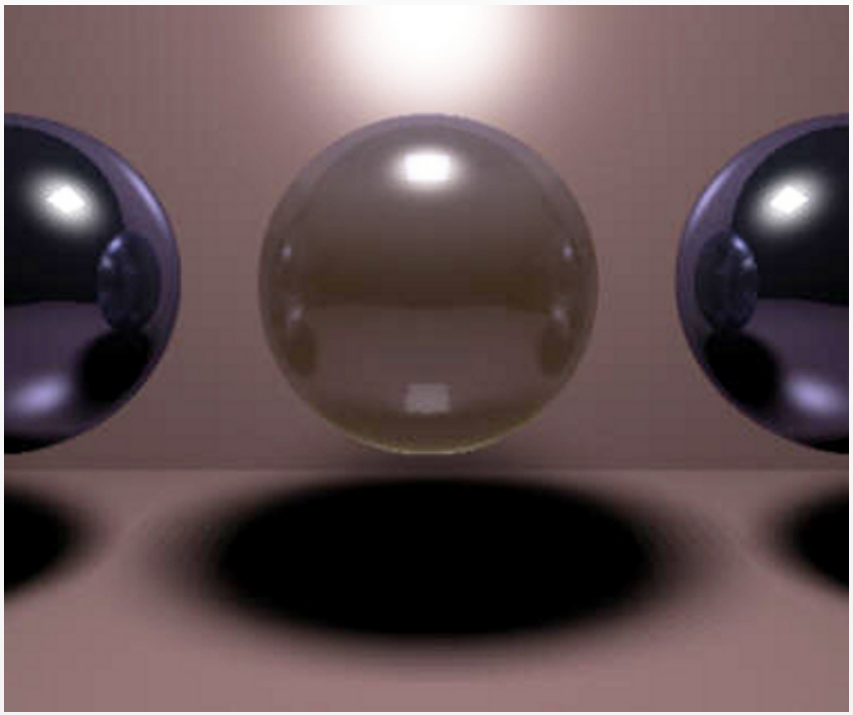
También puede usarse para simular la profundidad de campo

(imagen creada por Andreas Byström usando el software **Arnold**, de Marcos Fajardo)

Ray-tracing: sombras graduales (soft shadows)

Los bordes de sombras arrojadas en la realidad son graduales por la extensión de las fuentes de luz (no son puntos). Esto puede simularse también:

☞ http://www.devmaster.net/articles/raytracing_series/part5.php



Integrales multidimensionales

En ray tracing, para tener en cuenta todos estos efectos, y calcular la luz en cada pixel es necesario calcular una **integral multidimensional**. El número de dimensiones de esta integral es:

- 2 en el área de cada pixel (x, y)
- 1 en el eje del tiempo, para motion-blur (t)
- 2 en la lente, para profundidad de campo (u, v)
- 2 en la fuente de luz, para sombras graduales (s, t)

en total son 7 dimensiones, y podrían ser más si se tienen en cuenta otros efectos.

Integración numérica por métodos de Monte-Carlo

La función que se debe integrar no tiene una expresión analítica, solo puede ser evaluada computacionalmente en puntos determinados su dominio de 7 dimensiones:

- La solución más eficiente en tiempo de cálculo es usar **integración numérica** por **métodos de Monte-Carlo** (MMC), situando puntos de muestra en posiciones aleatorias del dominio
- Los MMC emplean **variables aleatorias** cuyo valor promedio es la integral que se quiere calcular (a estas VA se les llama **estimadores**
- Para calcular el promedio, se usan métodos de **muestro estadístico**
- El error asociado a los MMC es menor que la asociada a otros métodos para espacio de tantas dimensiones.

Integración numérica MMC

En general, en un dominio D con puntos de d dimensiones, queremos calcular la integral I de una función f

$$I = \int_D f(\mathbf{x})\, d\mu(\mathbf{x}) \quad \text{donde: } \mathbf{x} = (x_1, x_2, \ldots, x_d)$$

suponiendo que podemos evaluar f pero no hay una expresión analítica de dicha función, podemos definir una variable aleatoria X_n en D^n:

$$X_n(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \frac{1}{n} \sum_{i=1}^{n} \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} \quad \text{se cumple: } E(X_n) = I$$

donde cada $\mathbf{x}_i$ es una VA con valores en D (son puntos aleatorios), distrbuidos según la **función de densidad de probabilidad** p, de integral unidad en D.

Integración numérica por métodos de Monte-Carlo

La literatura de métodos de Monte-Carlo es muy amplia (se comenzaron a usar para diseñar centrales nucleares en los años 50).

- Una técnica buena es usar **secuencia de números quasi-aleatorios**, como por ejemplo la **secuencia de Hammersley**, ya que presentan propiedades de **baja discrepancia**
- También es adecuado usar **muestreo por importancias**, especialmente **muestreo por importancias múltiple** y otras variantes (todas estas técnicas tratan de seleccionar un p que minimize la **varianza** de la VA X_n, usando el conocimiento disponible *a priori* sobre f.

Sección **6**

Cálculo de iluminación global

Iluminación global

Este término se refiere al calculo de iluminación indirecta reflejada entre los objetos. Las técnicas de Iluminación Global permiten calcularla de forma físicamente correcta:

fotografía

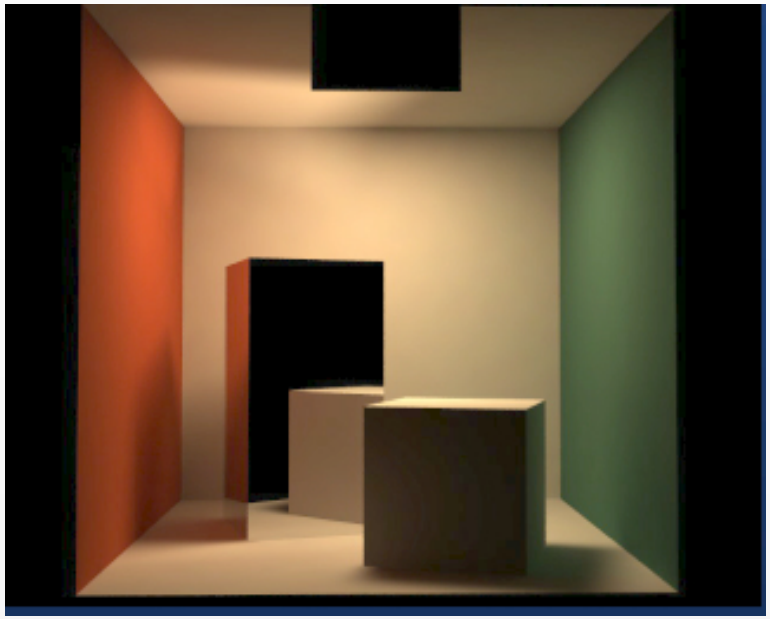


imagen generada

Iluminación global

La función que determina la iluminación emitida más reflejada en cada punto de las superficies (S) es una función f positiva, integrable y de cuadrado integrable, que cumple una **ecuación integral de Fredholm**, con la siguiente forma:

$$f(x) = g(x) + \int_{y \in S} k(x,y)\, f(y)\, dA(y)$$

donde:

- S es el dominio del problema (los puntos de las superficies del modelo)
- k es una función que se puede evaluar
- g es una función conocida (la iluminación emitida por las fuentes de luz).

Iluminación global: la ecuación integral

Esta ecuación se puede plantear como una **ecuación funcional** usando un **operador integral** $\mathcal{T}$ definido en un **espacio de Hilbert** de funciones integrables:

$$f \;=\; g + \mathcal{T} f$$

el operador $\mathcal{T}$ actua sobre una función cualquiera h, produciendo otra función $h' = \mathcal{T} h$, que se define por sus valores en un punto cualquier $x \in S$:

$$h'(x) \;=\; [\mathcal{T} h](x) \;=\; \int_{y \in S} k(x,y)\, h(y)\, dA(y)$$

Iluminación global

La ecuación anterior tiene solución (única) si la **norma** del operador $\mathcal{T}$ es inferior a la unidad, en ese caso, f se puede expresar como una **serie de Neumann** (por Carl Gottfried Neumann), como el límite de una serie de funciones:

$$f = \mathcal{T}^0 g + \mathcal{T}^1 g + \mathcal{T}^2 g + \ldots$$

esta función no puede expresarse analíticamente, y solo puede ser aproximada por métodos numéricos de dos clases:

- Usando **métodos de elementos finitos**
- Usando **métodos estocásticos o de Monte-Carlo**

Iluminación global: elementos finitos

La base de los métodos de elementos para ecuaciones integrales consiste en proyectar las funciones en un subespacio vectorial de funciones generados por **conjuntos finitos de funciones base**

- La ecuación integral se convierte en un sistema de ecuaciones (grande)
- Los sistemas se resuelven por métodos numéricos iterativos, como el método **Gauss-Seidel**, especialmente **métodos de relajación**, como la **relajación Southwell**
- Se puede mejorar la eficiencia usando funciones base multiresolución, como los **Wavelets**

Iluminación global: métodos de Monte-Carlo

los métodos de elementos finitos son poco eficientes en escenas complejas, se ha impuesto el uso de métodos de Monte-Carlo, mucho más eficientes en estos casos:

- Se usan estimadores basados en **cadenas de Markov** (o **paseos aleatorios**), que son secuencias finitas de variables aleatorias donde cada una solo depende de la anterior.
- Cada VA de la cadenas está definida sobre un punto del dominio y tiene valores reales. Una cadena es, por tanto, una VA definida sobre las posibles secuencias finitas de puntos del dominio.
- Es necesario usar PDFs para la primera variable aleatoria (p_0), para la transición de una a la siguiente (p_t), y para la probabilidad de terminar p_a.

Propiedades de las cadenas de Markov

Para un $z \in S$ determinado, se define la VA X_z como:

$$X_z(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \;=\; g(z) \;+\; \frac{k(x,y)}{p_t(x,y)}\, X_{\mathbf{x}_0}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$$

(nótese que es una definición recurrente). Supuesto que se cumplen estos requisitos:

$$p_0(x) = p_t(z,x) \qquad k(x,y) > 0 \;\rightarrow\; p_t(x,y) > 0 \qquad p_a(x) > 0$$

entonces:

$$E(X_z) \;=\; f(z)$$

con lo cual, muestreando aleatoriamente muchas cadenas podemos aproximar bien los valores de f en un punto z determinado.

Aplicaciones de los métodos MMC

Se usan sobre todo en aplicaciones de marketing inmobiliario y películas generadas por ordenador. Requiere de un tiempo de cálculo elevado (por ejemplo, para cine generado por ordenador, se necesitan varias horas por cada cuadro o frame de la película).

- Los estudios necesitan alquilar mucho de tiempo de cálculo a empresas que poseen grandes sistemas con cientos de ordenadores (*clusters*).
- Se pueden diseñar estimadores de varianza reducida que necesitan un menor número de cadenas por pixel, y obtienen la misma calidad en menos tiempo.
- El resultado es que **las buenas matemáticas pueden ahorrar dinero a las empresas.**

Ejemplos de iluminación global para rendering realista

(creado por Greg Ward en 1994, usando su software *Radiance*)

Ejemplos de iluminación global para rendering realista

(creado por Henrik Van Jensen en 2000, usando su software *Dali*)

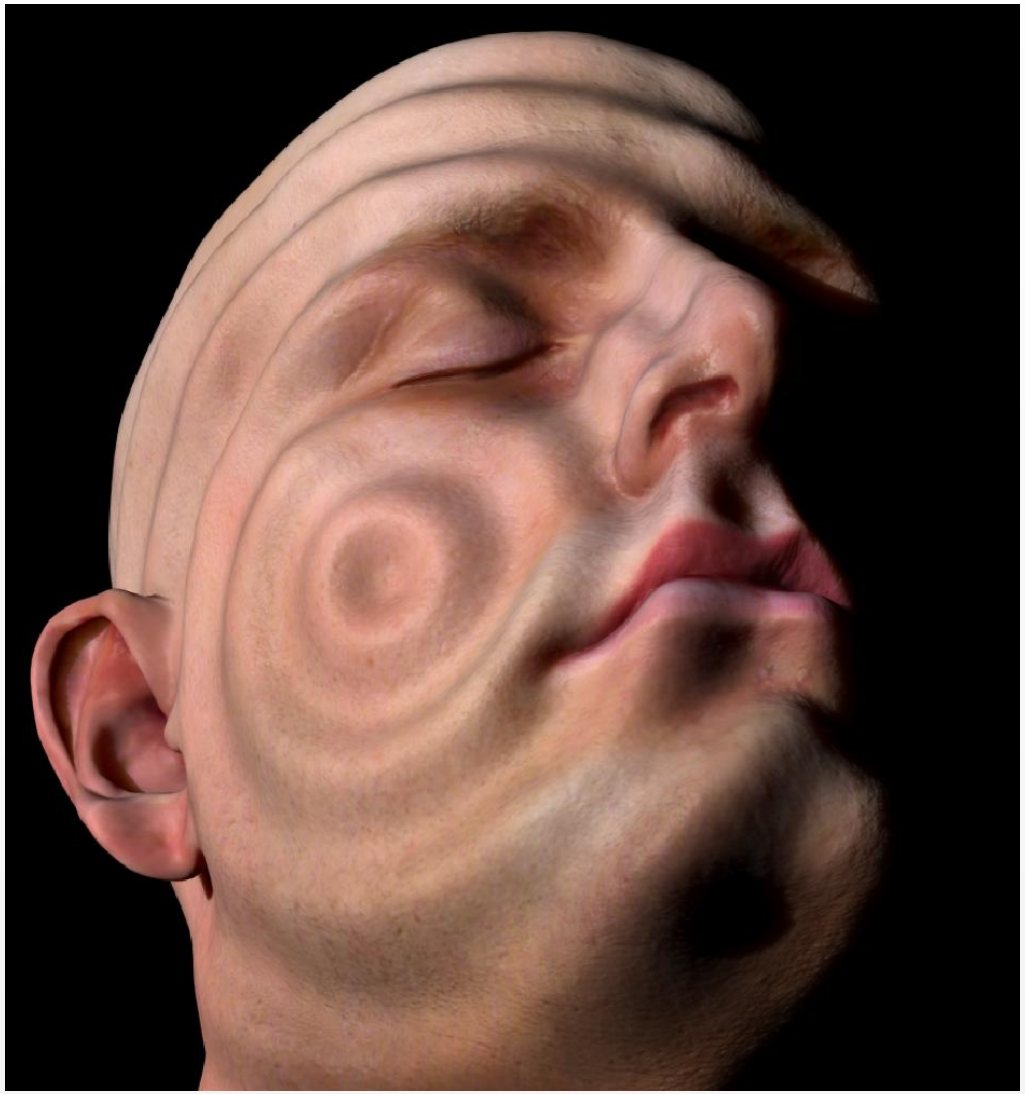
Ejemplos de iluminación global para rendering realista

(creado por Andreas Byström usando el software *Arnold* de Marcos Fajardo)

Ejemplos de iluminación global para rendering realista

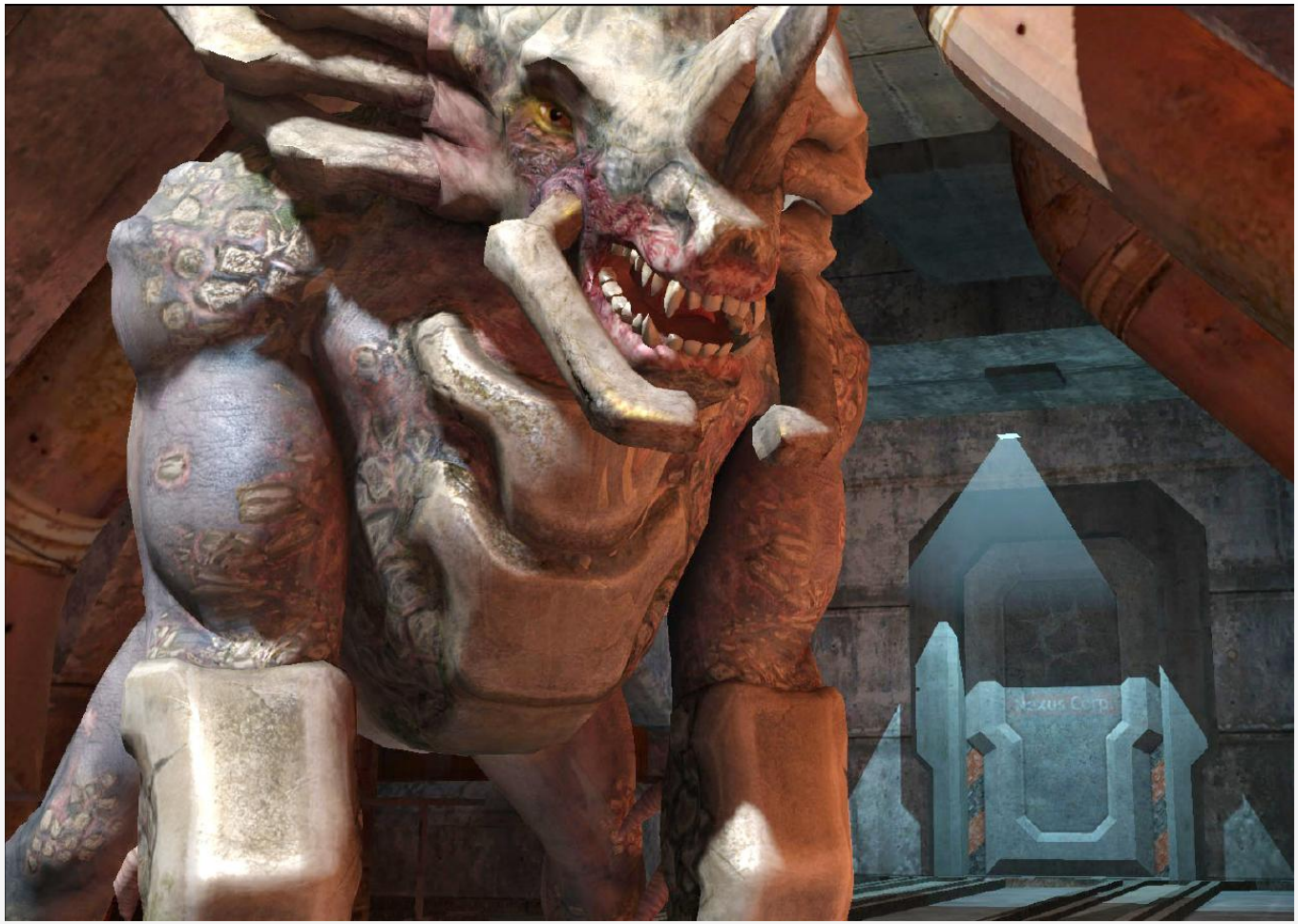
(creado por Eugen D'Eon y otros)

Ejemplos de iluminación global para rendering realista



(creado por Eugen D'Eon y otros)

Iluminación global para videojuegos



(creado por McGuire y otros)

Sección 7

IG en la Universidad de Granada

IG en la Universidad de Granada

El Grupo de Investigación en Informática Gráfica (del Departamento de Lenguajes y Sistemas Informáticos) trabaja, entre otros, en estos campos:

- Rendering: producción de imágenes realistas a partir de modelos
- Non-photorealistic rendering : producción de imágenes no realistas a partir de modelos o fotografías
- Adquisición de modelos: obtención de modelos mediante escáner láser
- Realidad Virtual: sistemas interactivos e inmersivos de visualización de modelos o entornos

Docencia

Docencia de pregrado en:

- Grado en Ingeniería Informática
- Ingeniería en Informática
- Ingenierías Informática de Sistemas y Gestión

Docencia/investigación de posgrado en:

- Máster en Universitario en Desarrollo de Software (MDS)
- Programa de Doctorado en Tecnologías de la Información y la Comunicación

Asignaturas de IG en MDS

Asignaturas impartidas por miembros del grupo de investigación:

- Digitalización 3D
- Fundamentos de Geometría y Geometría Computacional.
- Modelado y Visualización de Volúmenes
- Programación Gráfica de Altas Prestaciones
- Realidad Virtual
- Realismo e Iluminación Global
- Técnicas Avanzadas de Modelado de Sólidos.
- Visualización Expresiva y Animación

más información del máster en:

☞ http://posgrados.ugr.es/master-desarrollo-software