Computación Gráfica = CG = Computer Graphics

¿Porqué computación gráfica? → *Necesidad* contemporanea
Visualizar/*Analizar* lo imposible
Una imagen vale más que mil palabras

¿Para qué sirve? → Representar visualmente información digital

¿Cómo se utiliza? → Lenguajes de programación, softwares, API's, Librerías, ...

¿Cuáles son las principales áreas? → Imágenes, Videos, Modelado 3d, captura, digitalización, animación, GPU, geometría computacional, Real-time rendering,

SIGGRAPH (ACM)

[International Conference on Computer Graphics and Interactive Techniques]

Figure 1: Our algorithm allows us to precompute detailed boundary layer data and efficiently reuse it for new simulations. We are able to generates turbulent vortices taking into account the relative velocity of an obstacle in the flow. Here, we apply our algorithm to a very thin object that is barely represented on the simulation grid.

©Blender Foundation & Mammoth HD

Figure 1: *Two examples displaying results from our interactive framework for video retargeting. The still images from the animated short "Big Buck Bunny" compare the original with the retargeted one. The pictures on the right show two different rescales. Thanks to our interactive constraint editing, we can preserve the shape and position of important scene objects even under extreme rescalings.*

SIGGRAPH (ACM)

[International Conference on Computer Graphics and Interactive Techniques]

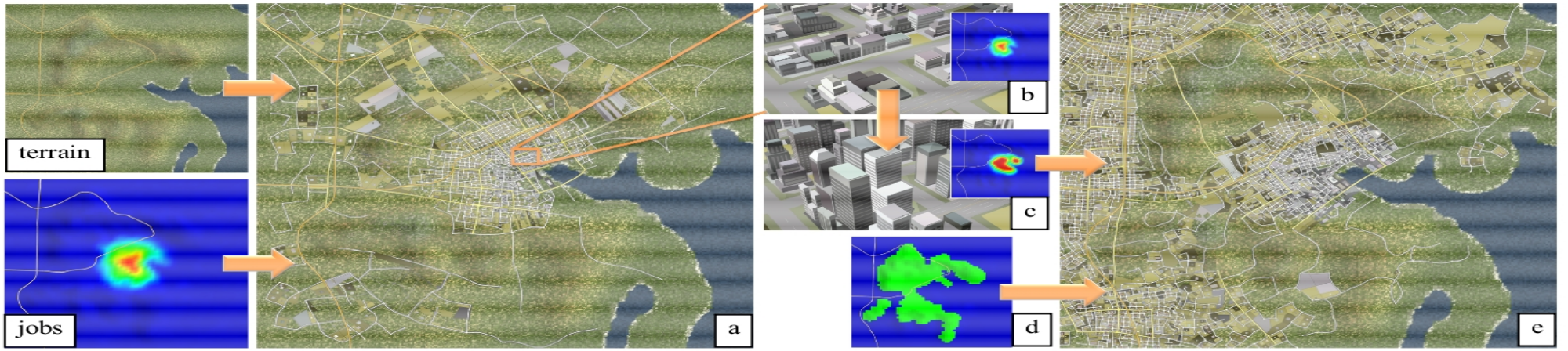


Figure 1. Urban Model Design. *This example city is incrementally generated in a two-step process. First, based on designer input, the system creates a low-density town in a valley by the coast (a). Then, the designer replaces the office buildings (b) with high-rises (c), and constrains the downtown of the existing city and the forest area around it (d). The system increases the population as a result of the larger number of jobs and locates the population in accessible land outside the valley, creating new roads, parcels, and buildings, while leaving the original downtown unchanged (e, f).*

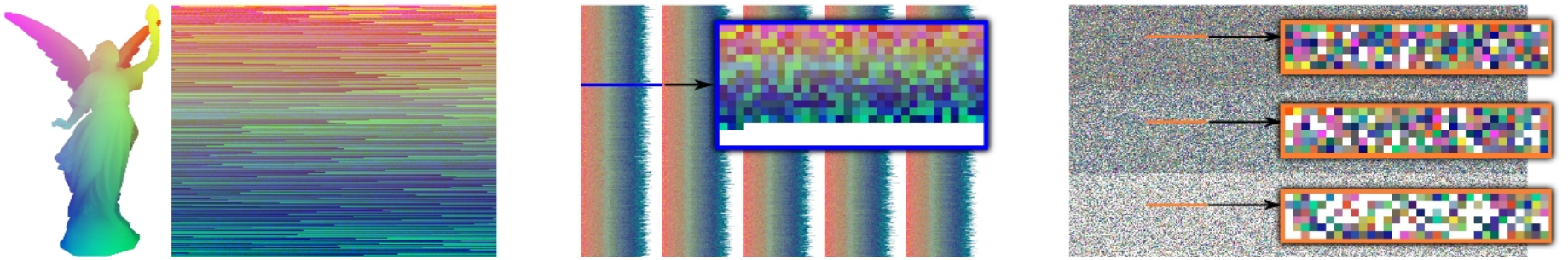


Figure 1: *Overview of our construction for a voxelized Lucy model, colored by mapping x, y, and z coordinates to red, green, and blue respectively (far left). The 3.5 million voxels (left) are input as 32-bit keys and placed into buckets of $\leq$ 512 items, averaging 409 each (center). Each bucket then builds a cuckoo hash with three sub-tables and stores them in a larger structure with 5 million entries (right). Close-ups follow the progress of a single bucket, showing the keys allocated to it (center; the bucket is linear and wraps around left to right) and each of its completed cuckoo sub-tables (right). Finding any key requires checking only three possible locations.*

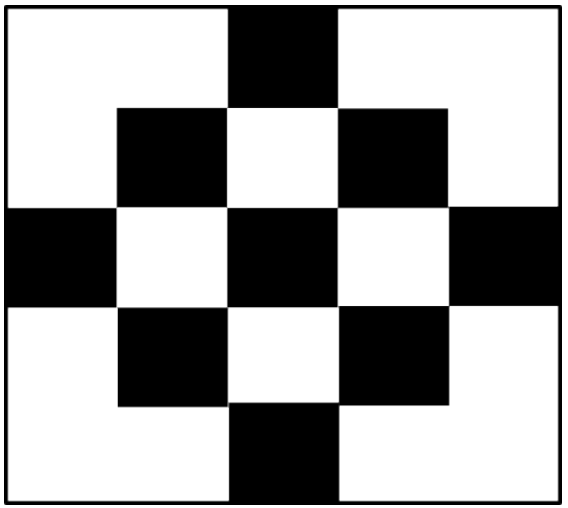
Conceptos Básicos

Modelo binario (Bitmap)

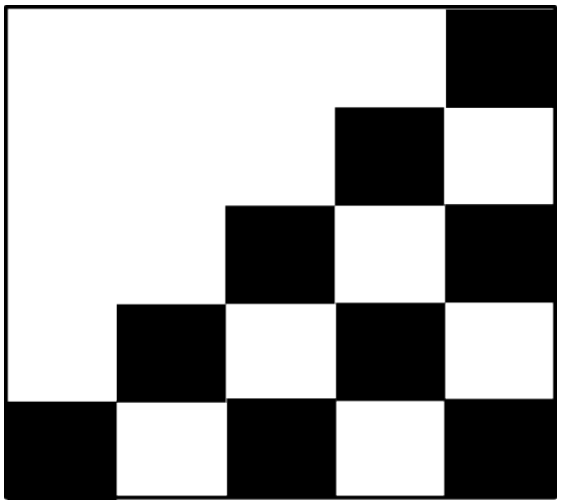
A=[0010001010101010101000100]

B=[0000100010001010101010101]

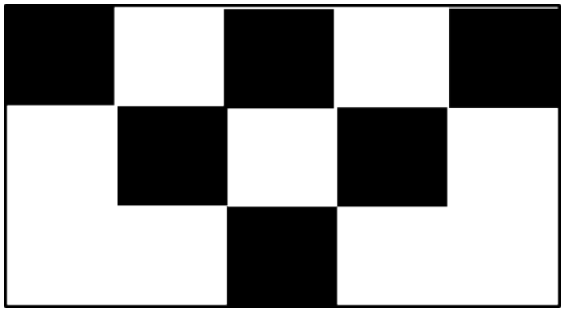
C=[101010101000100]



A



B



C

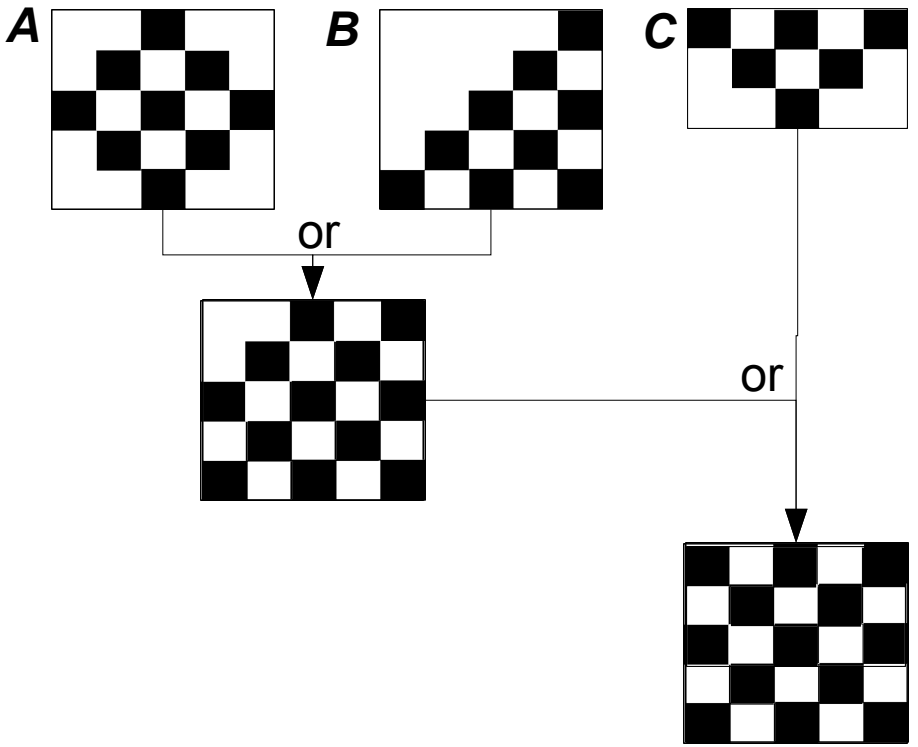
Conceptos Básicos

Modelo binario (Bitmap)

A or B = [00100010101010101010000100] or [00001000100010101010101010101]

= [00101010101010101010101010101]

A or B or C = [10101010101010101010101010101]



Conceptos Básicos

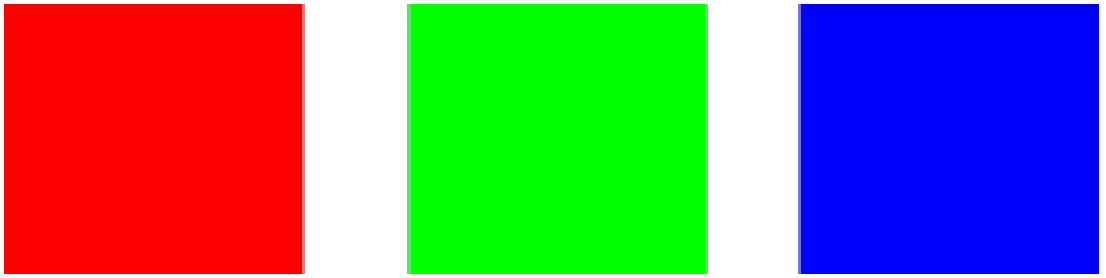
Espacios de colores RGB

$$C = [C_{red}, C_{green}, C_{blue}]$$

$$C_{red} \in \{0, 1\}$$

$$C_{green} \in \{0, 1\}$$

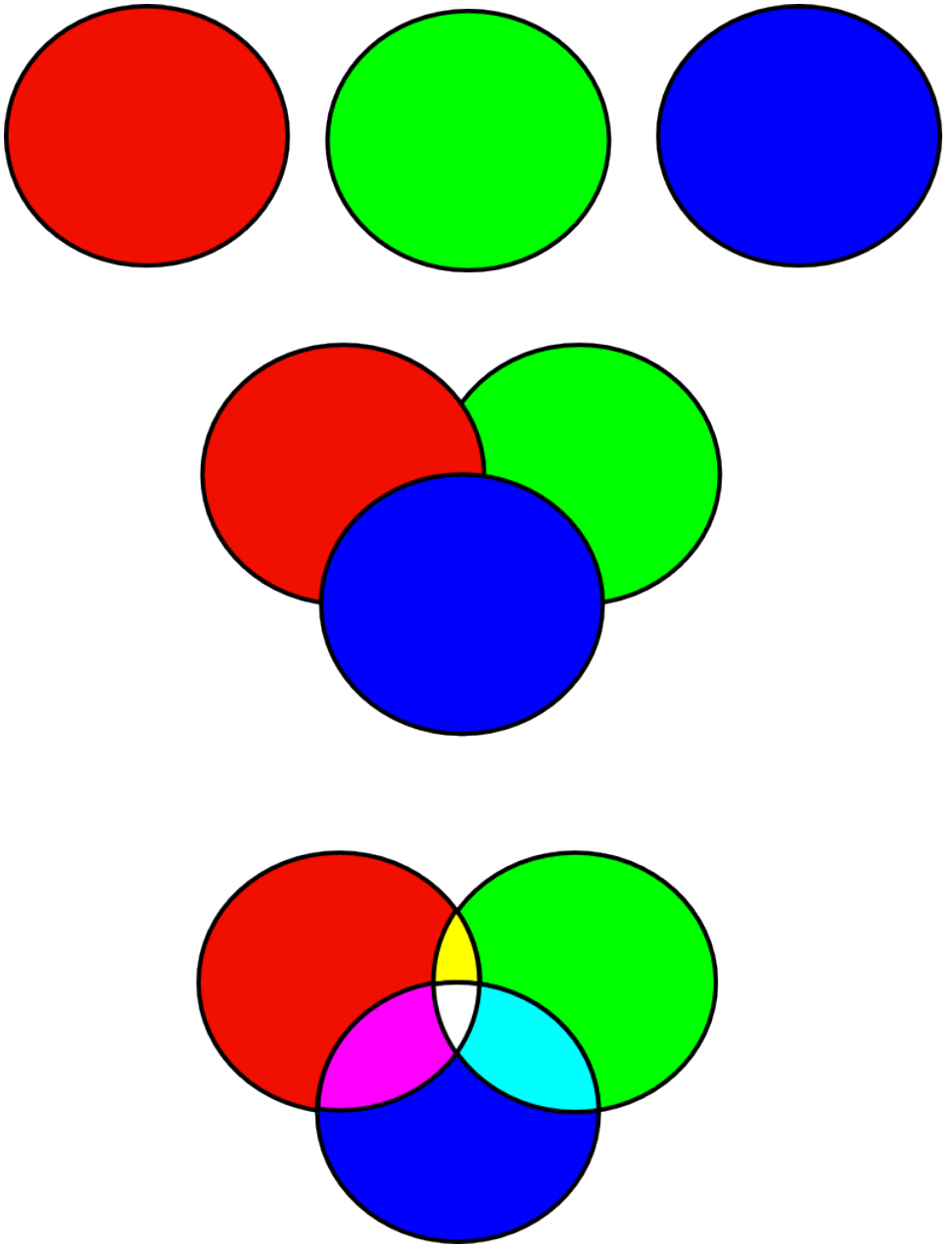
$$C_{blue} \in \{0, 1\}$$



1 1 1

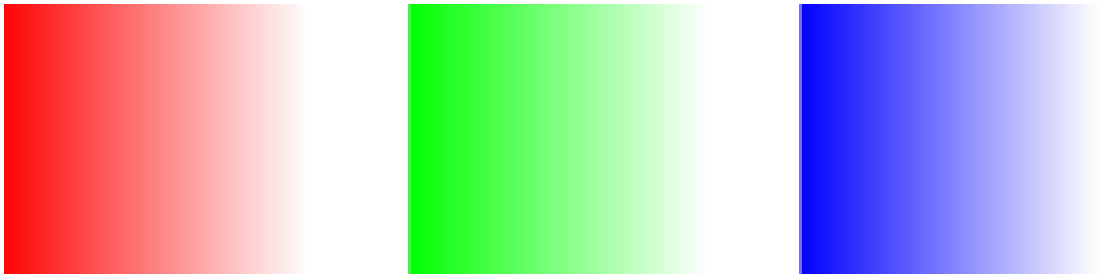
Conceptos Básicos

Espacios de colores RGB



Conceptos Básicos

Espacios de colores RGB



8 bits = 1 byte $\longrightarrow [0, 255]$

16 bits = 2 bytes = 1 Small $\longrightarrow [0, 65535]$

32 bits = 4 bytes = 1 Integer $\longrightarrow [0, 4294967295]$

64 bits = 8 bytes = 1 Float $\longrightarrow [0, 18446744073709551615]$

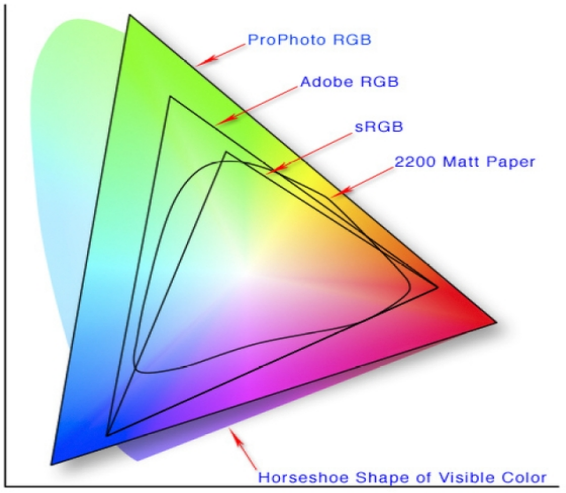
$$256 \times 256 = 65536$$
$$256 \times 256 \times 256 \times 256 = 4294967296$$
$$256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 \times 256 = 18446744073709551616$$

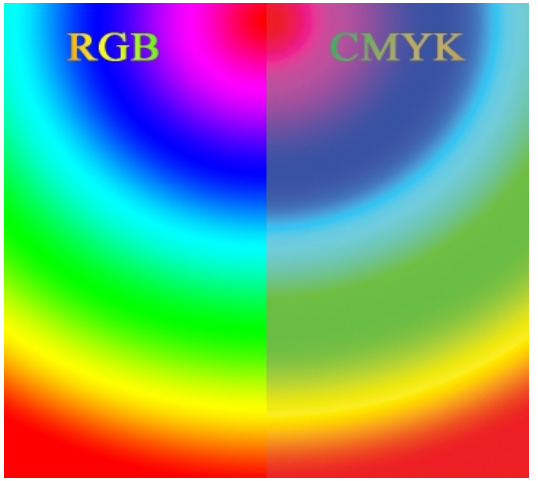
Conceptos Básicos

Otros espacios de colores

HSV **HSL** **YcbCr** **CMYK**

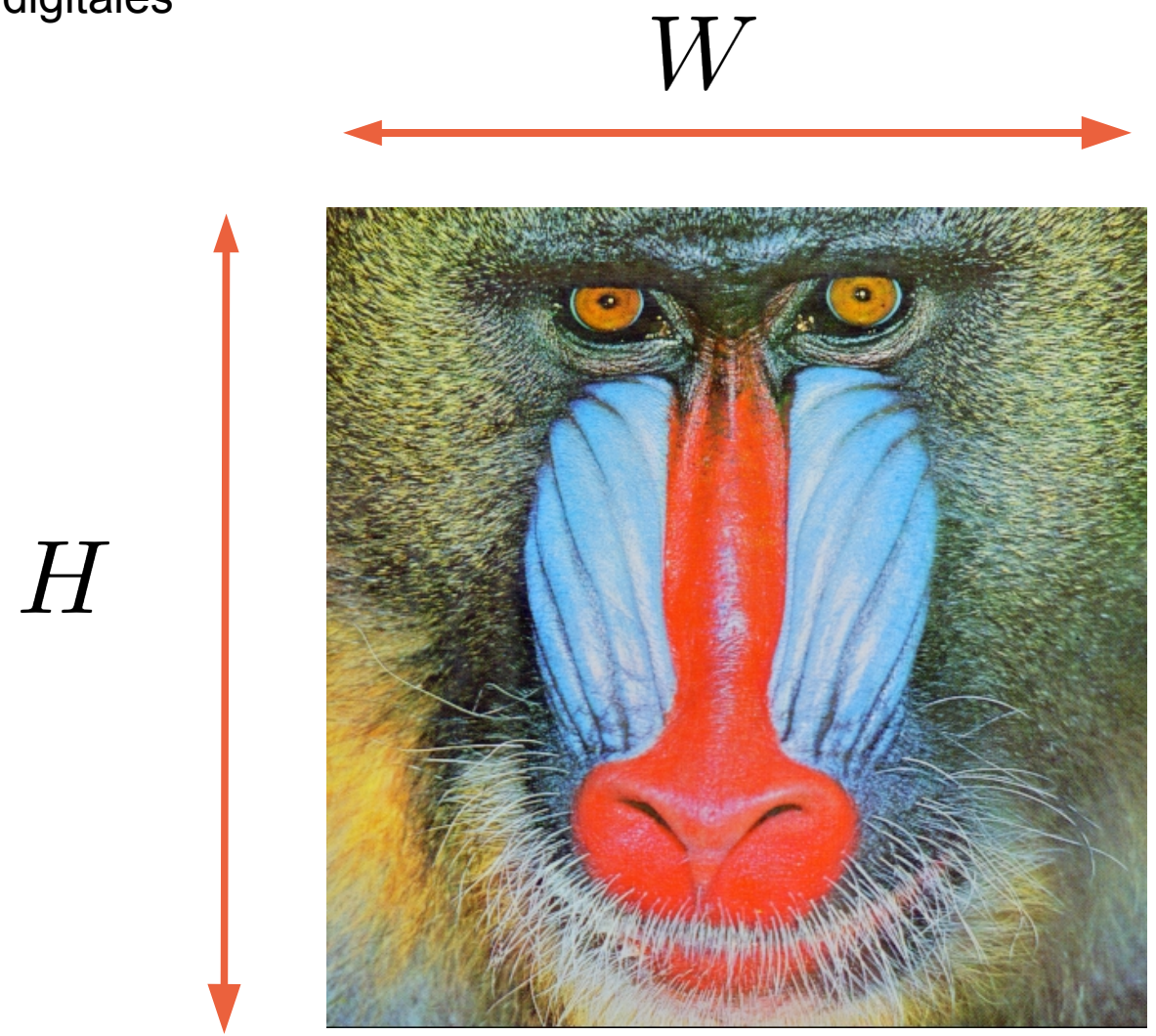
Cada aplicación tiene uno o varios espacios que se adecúan mejor a la misma





Conceptos Básicos

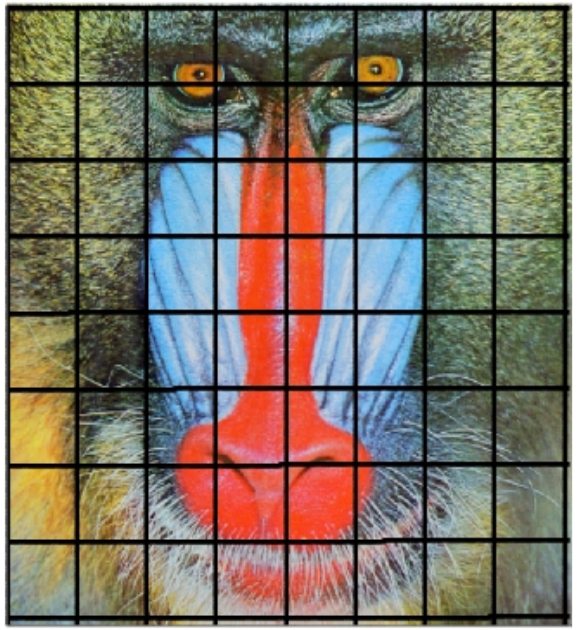
Imágenes digitales



Conceptos Básicos

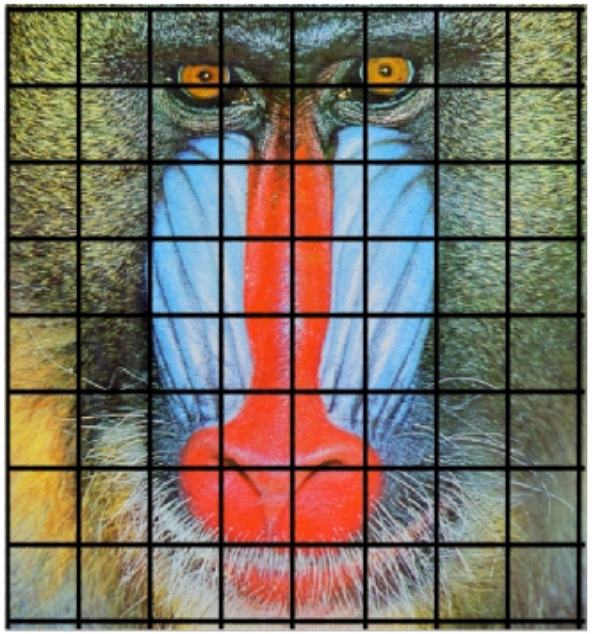
Imágenes digitales

$(0,0)$



$(W-1,H-1)$

$(W-1,H-1)$



$(0,0)$

Conceptos Básicos

Imágenes digitales: **Arreglos**

$$Ch_1 = \left(\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,W-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,W-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{H-1,0} & a_{H-1,1} & \cdots & a_{H-1,W-1} \end{bmatrix} \right)$$

$$Ch_2 = \left(\begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,W-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,W-1} \\ \cdots & \cdots & \cdots & \cdots \\ b_{H-1,0} & b_{H-1,1} & \cdots & b_{H-1,W-1} \end{bmatrix} \right)$$

$$Ch_3 = \left(\begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,W-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,W-1} \\ \cdots & \cdots & \cdots & \cdots \\ c_{H-1,0} & c_{H-1,1} & \cdots & c_{H-1,W-1} \end{bmatrix} \right)$$

$$Ch_4 = \left(\begin{bmatrix} d_{0,0} & d_{0,1} & \cdots & d_{0,W-1} \\ d_{1,0} & d_{1,1} & \cdots & d_{1,W-1} \\ \cdots & \cdots & \cdots & \cdots \\ d_{H-1,0} & d_{H-1,1} & \cdots & d_{H-1,W-1} \end{bmatrix} \right)$$

$$I := [Ch_1, Ch_2, Ch_3, Ch_4,]$$

Conceptos Básicos

Imágenes digitales: **Arreglos**

Estructuras tipo arreglos (ordenados)

Memoria lineal en las dimensiones = (WxH)

Operaciones Cuadráticos en las dimensiones $(WxH)^2$

Conceptos Básicos

Imágenes digitales: **Funciones**

$$I : \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{C}$$

$$\mathbb{C} := \{(x, y, z) : x, y, z \in \mathbb{R} \vee \mathbb{Z}\}$$

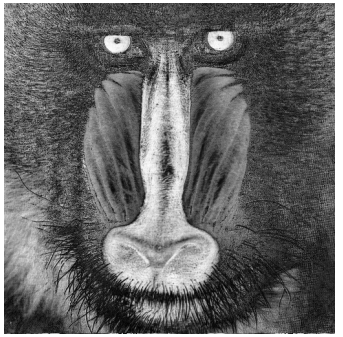
$$T\Big(I\Big) : I \mapsto I$$

$$\int dx, \partial x, \int \int dxdy, \partial xy$$

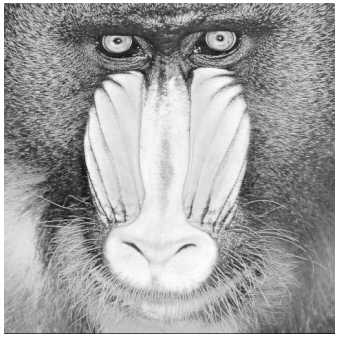
Conceptos Básicos

Imágenes digitales

H

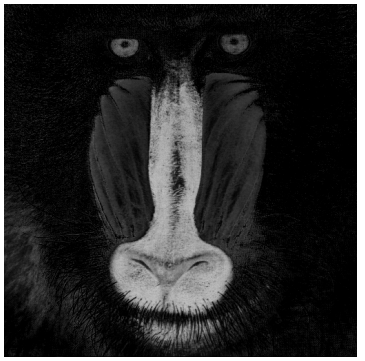


S

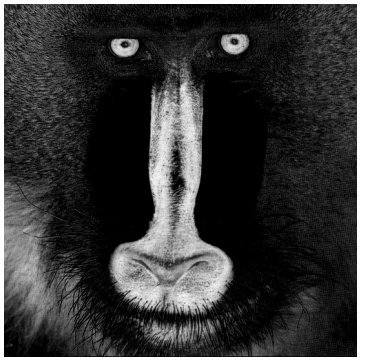


V

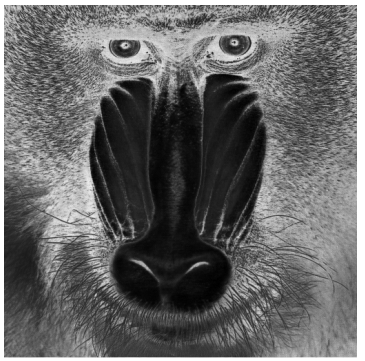
C



M



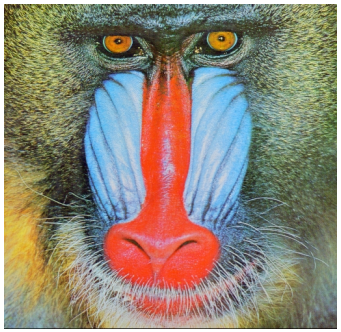
Y

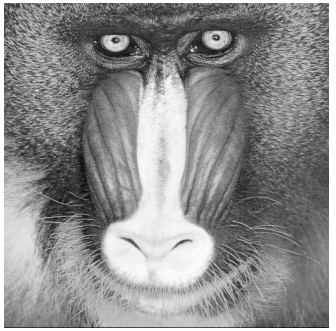


K

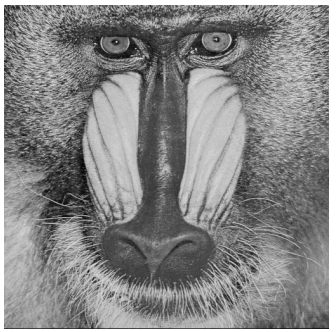
Conceptos Básicos

Imágenes digitales

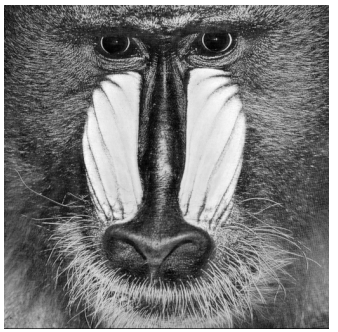




R

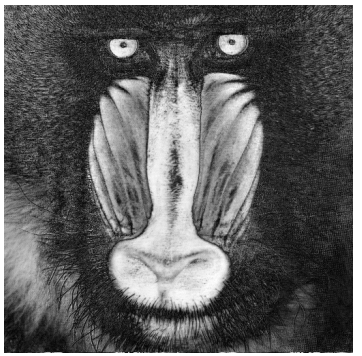


G

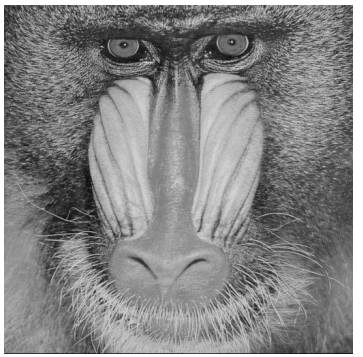


B

H



S

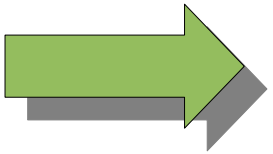


L

Conceptos Básicos

Escritura compresión

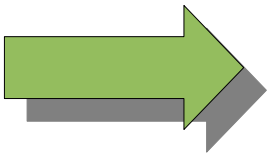
1. Buscar repeticiones
2. Indexarlas
3. Guardar



Pérdida

Formatos con compresión (con pérdida)

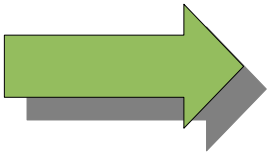
1. Comprimir
2. Guardar cabeceras
3. Guardar cada canal



Escribe menos
Pesa menos
Procesa más
Pierde de calidad

Formatos sin compresión (sin pérdida)

1. Guardar cabeceras
2. Guardar cada canal



Pesa más
Procesa menos
Escribe más
Mantiene calidad

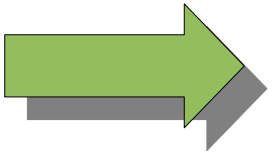
Conceptos Básicos

Lectura de compresión

1. Leer cabeceras
2. Leer bloque
3. Usar diccionario
4. Reconstruir componente de imagen

Lectura de Formatos con compresión (con pérdida)

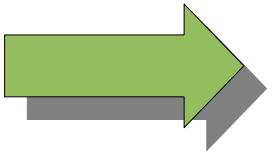
1. Lectura de compresión
2. Disponer en estructura (arreglo, listas, etc)



- Lee menos
- Procesa más
- Usa menos memoria

Formatos sin compresión (sin pérdida)

1. Leer cabecera
2. Leer bloques
3. Disponer en estructura (arreglos, listas, etc)



- Lee más
- Procesa menos
- Usa más memoria

Conceptos Básicos

Resolución

Bytes por color (gamma de colores)

Bytes por pixeles (gamma de colores)

Pixeles por imagen (tamaño en display)

Pixeles por cm (tamaño de impresión)

Crecimiento (en tamaño) ~ Resolución ~ Calidad

Calidad <= Calidad (Resolución de captura)

Conceptos Básicos

Videos digitales

$$I : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{C}$$

$$x, y, t \mapsto c \in \mathbb{C}$$

(Stacks de imágenes)

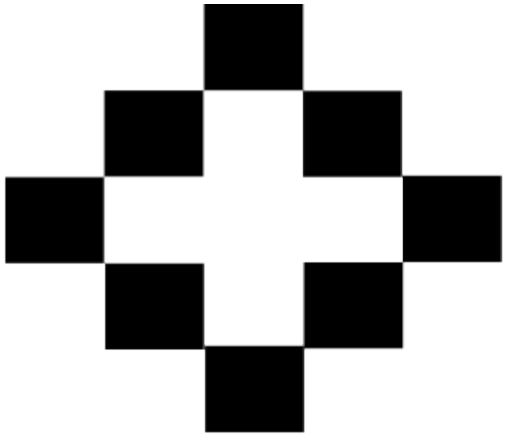
Estructuras de imágenes enlazadas (secuencias)

Más memoria (lineal en las dimensiones = WxHxT)

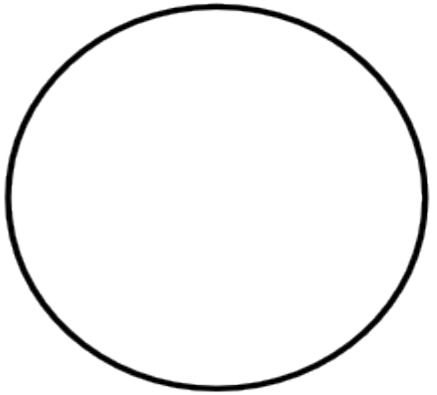
Operaciones Cuadráticos en las dimensiones Lx(WxH)^2

Conceptos Básicos

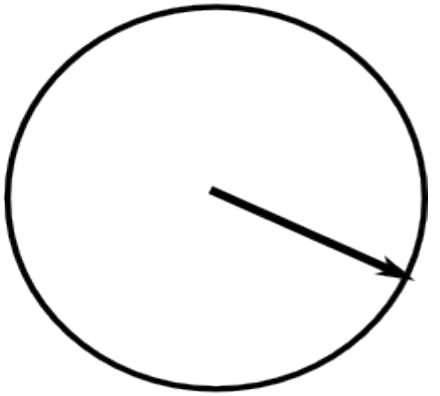
Formas (2d)



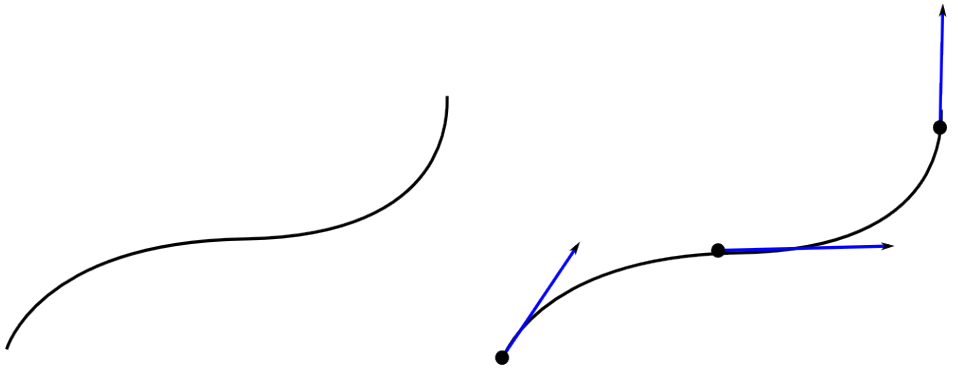
[0010001010101010101000100]

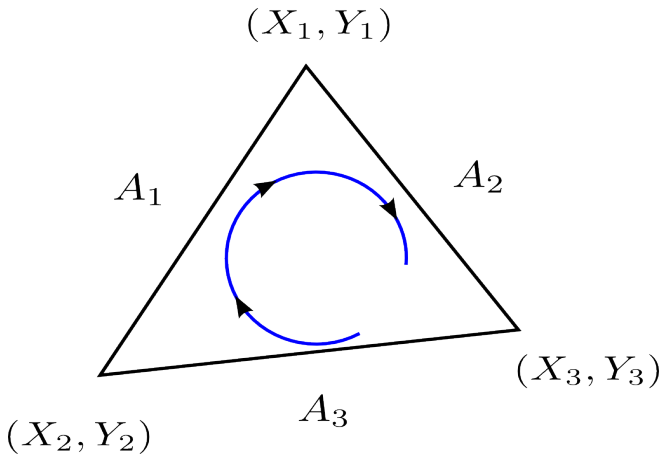


$f(x,y) = [+-]\sqrt{x^2+y^2}$



$(R,(x,y))$





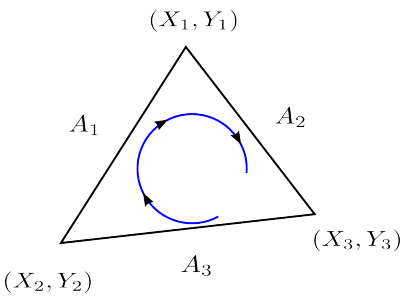
Conceptos Básicos

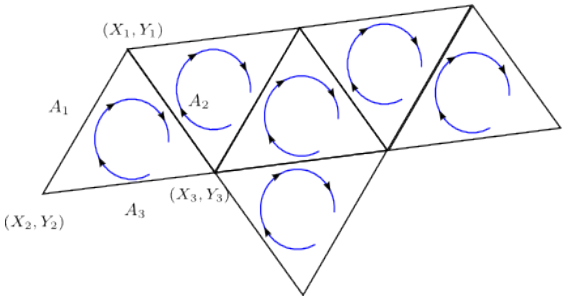
Formas (3d)

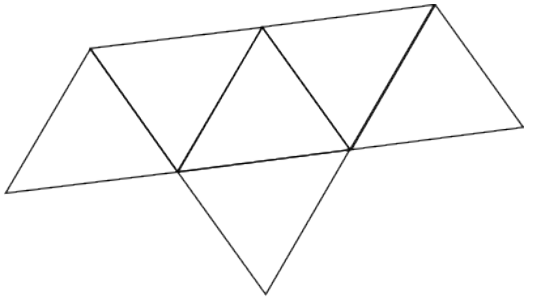
Coordenadas	
Puntos	N-tupla de coordenadas (N:=1,2,3,4)
Vértice (vertex)	Puntos
Aristas (edge)	Secuencia ordenada de vértices
Triángulos (triangle)	Secuencia ordenada de aristas/vértices o ambas
Superficies (surface)	Secuencia ordenada de triángulos
Volumenes (volume)	Secuencia ordenada de superficies

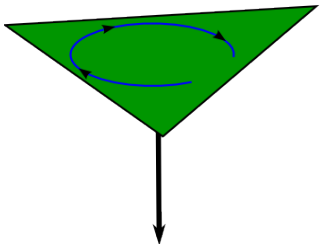
Conceptos Básicos

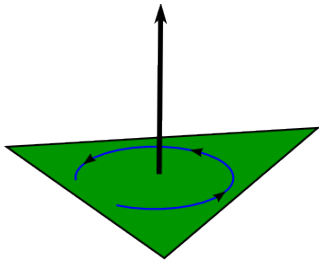
Formas (3d)

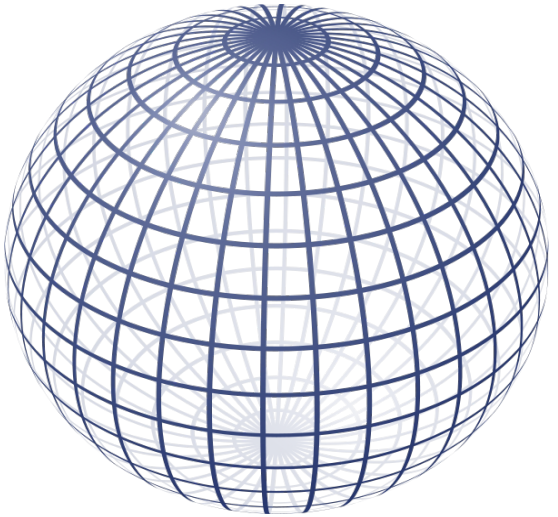






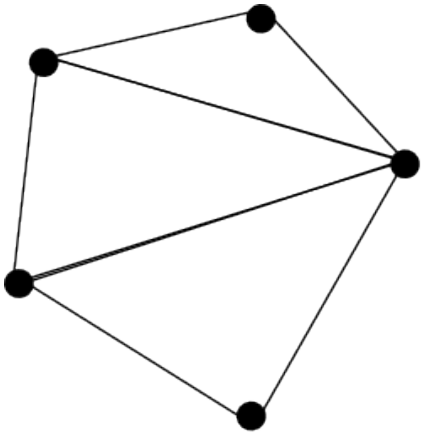


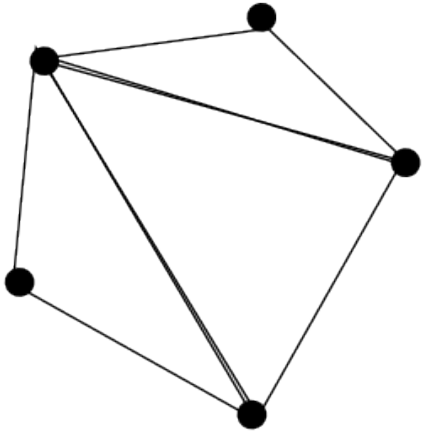


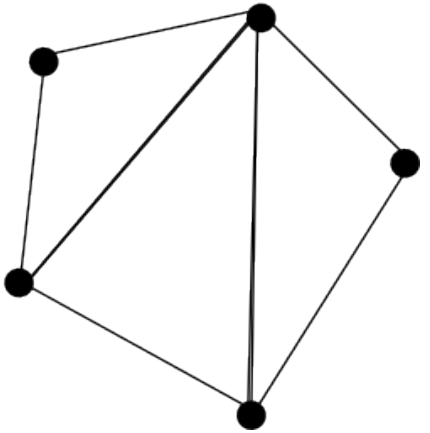


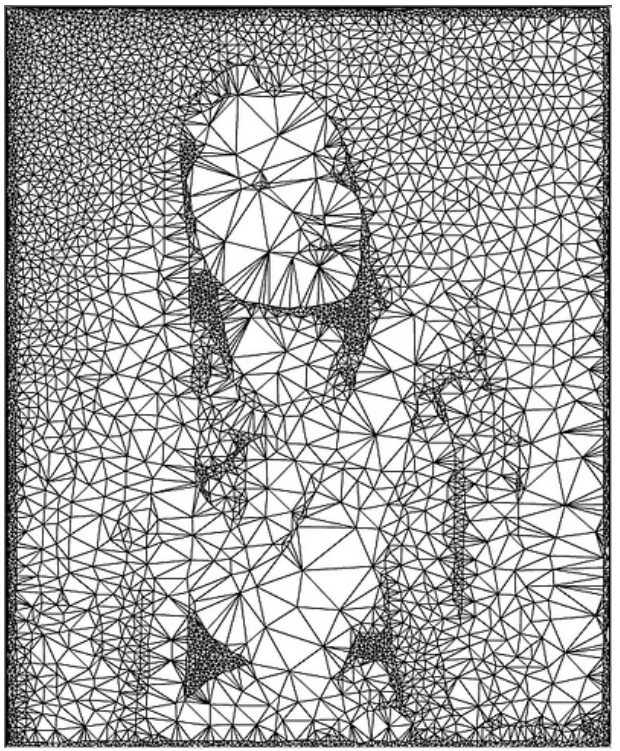
Conceptos Básicos

Triangulaciones (Triangulation/Facets 3d)



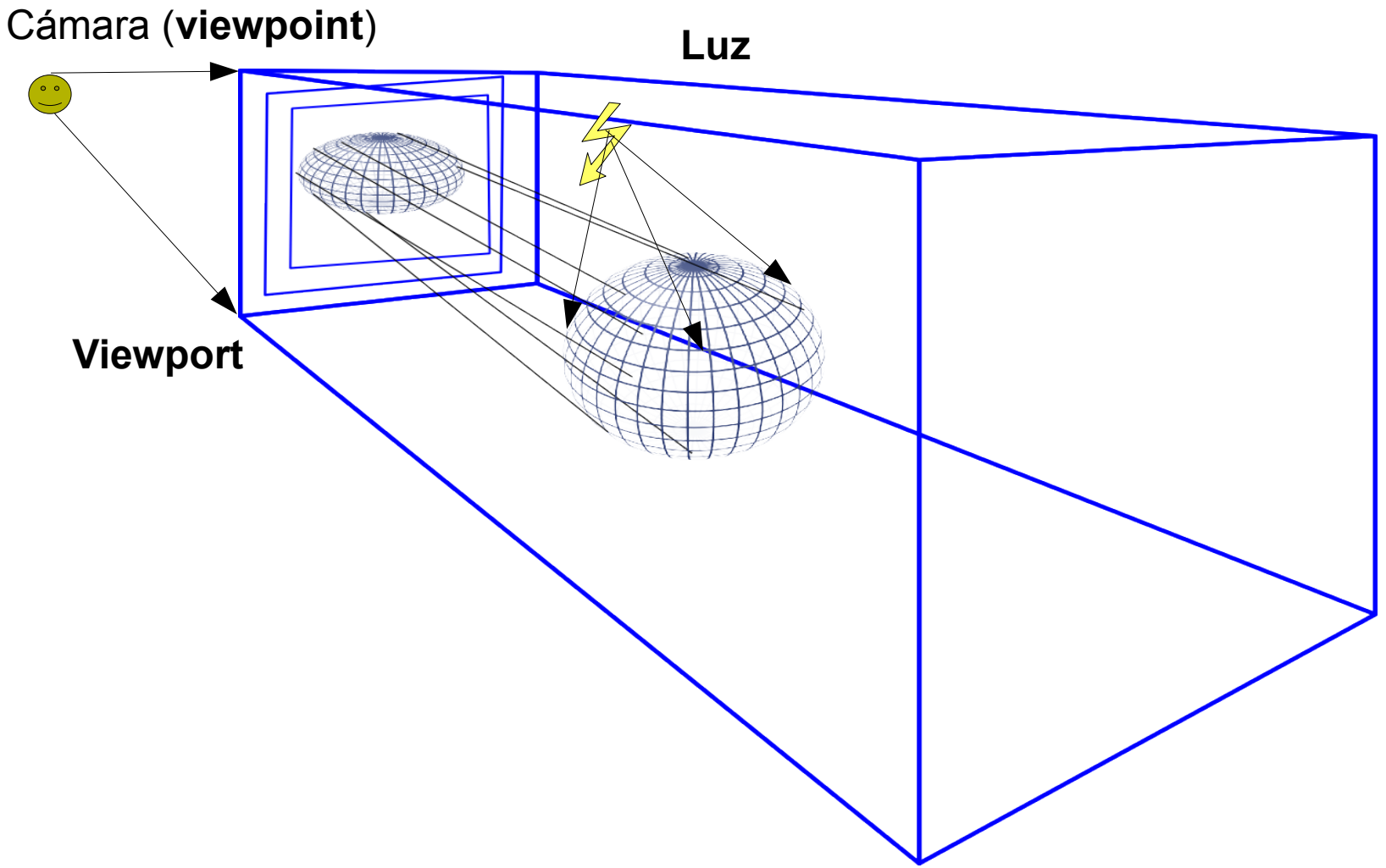






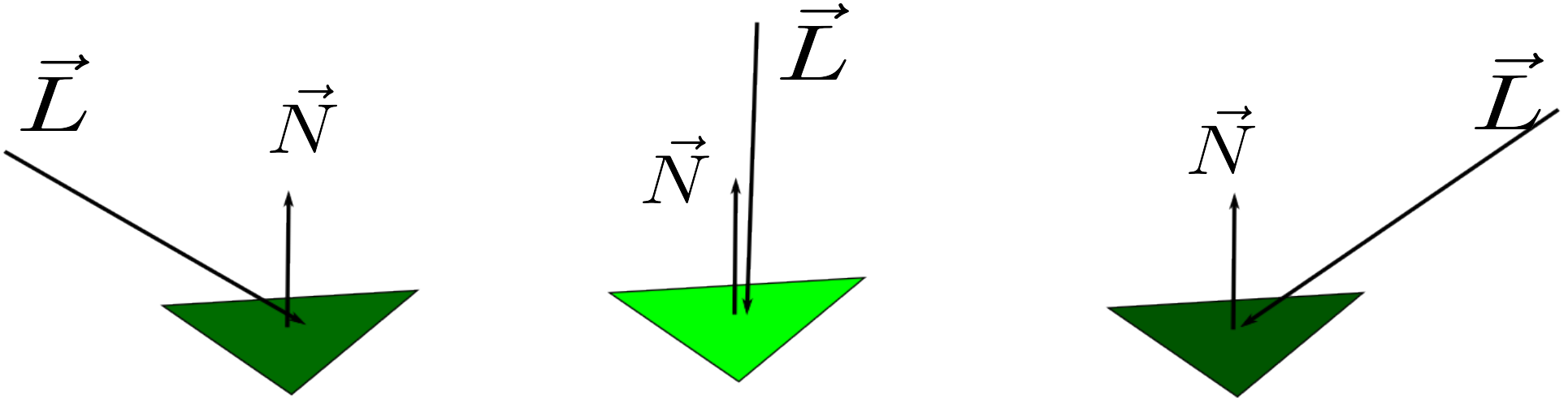
Conceptos Básicos

Proyecciones



Conceptos Básicos

Proyecciones



$$\vec{L} \cdot \vec{N}$$

Herramientas

- Autocad / *Cad / 3DS-Max / Inventor / Maya / Softimage
- GnuPlot
- Matlab / Octave (Maximma)
- Java2D + Toolkit's (AWT, Swing, SWT, GLUT, Flex, GWT, QT, GTK+)
- Java3D
- OpenGL (Java, C++, Python)
- OpenCV (C++, Python)
- Cuda
- Cg (Nvidia)
- ARB

Herramientas

Esquema de trabajo Java 2d

1. Crear primitiva de ventana V.
2. Configurar V.
3. Crear primitiva de dibujo G.
4. Configurar G.
5. Dibujar usando primitivas de G.
6. Desplegar.

Esquema de trabajo API's 3d

1. Configurar coordenadas espacio S.
2. Fijar cámaras C (puntos de vista) en S.
3. Fijar luces L.
4. Dibujar usando primitivas en S (opcional).
5. Leer archivos de primitivas A sobre S (opcional).
6. Render (procesar).

Herramientas

Esquema de trabajo genérico

1. Configurar modo despliegue (display) D.
2. Dibujar sobre D.
3. Desplegar D.

Ejercicio

Implementar la función *gradiente* que recibe un arreglo de dimensiones WxHx4, representando una imagen de 4 canales: R,G,B y alpha (transparencia). Se espera que esta función modifique la imagen para obtener un gradiente de colores Verdes, como el ejemplo de la figura 1. El canal de transparencia es proporcional a la intensidad del color.

```
public void gradiente(int[][][] image)
{


}
```

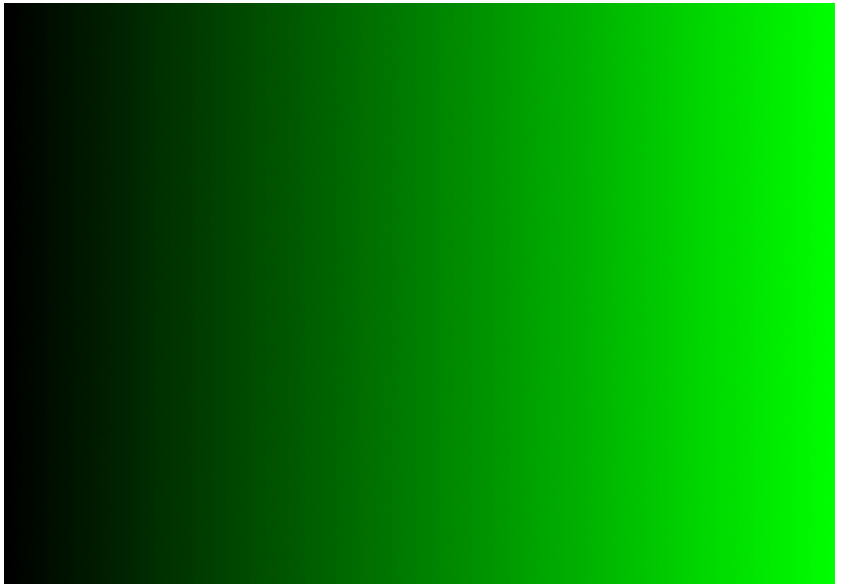


Figura 1