Universidad de Huelva

Tema 2

Informática gráfica y OpenGL

Índice

2.1 Informática gráfica

2.2 Principios básicos de representación en 3D

2.3 Introducción a OpenGL

2.4 La evolución de OpenGL

2.5 Aspectos básicos de OpenGL

2.1 Informática gráfica

2.2 Principios básicos de representación en 3D

2.3 Introducción a OpenGL

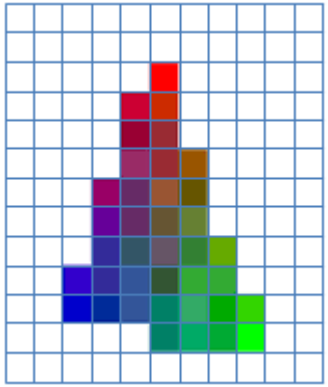
2.4 La evolución de OpenGL

2.5 Aspectos básicos de OpenGL

- La **Informática gráfica** (***Computer graphics***) es la rama de la Informática dedicada a la generación y manipulación de imágenes.
- El término fue propuesto por primera vez en 1960 por William Fetter, diseñador gráfico de Boeing.
- Durante la década de 1960's se desarrollaron las primeras técnicas de generación de imágenes:
  - En 1962, Steve Russell desarrolló *Spacewar!*, el primer videojuego publicado con éxito en el mundo académico. Unos años antes se había programado *Tennis for two*, pero no tuvo ninguna repercusión.
  - A principios de los 60's Pierre Bèzier, ingeniero de Renault, propuso la formulación matemática de curvas para modelado 3D.

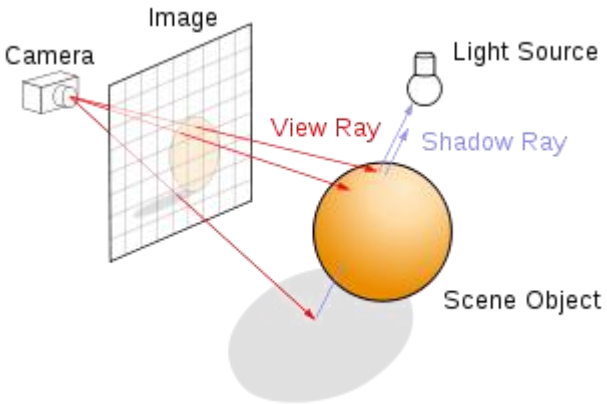
- A mediados de los 60's la Universidad de Utah se convierte en el principal centro de investigación en esta materia. Los investigadores de esta universidad desarrollaron, entre otras técnicas, los sombreados Gouraud, Phong, y Blinn, los mapas de texturas, algoritmos de determinación de cara oculta, subdivisión de superficies, trazado de líneas en tiempo real y los primeros intentos de realidad virtual.
- En 1969 se creó el Grupo de Interés en Informática Gráfica de la ACM (SIGGRAPH). Desde 1974 este grupo organiza el congreso internacional SIGGRAPH, que es la referencia mundial en el desarrollo de esta materia.

- **Raster**: Se denomina raster a la representación de una imagen como una matriz de píxeles.



- **Pixel**: Acrónimo de picture element. Cada uno de los puntos de color que se muestran en una pantalla.
- **Gráficos vectoriales**: Formato de imagen formado por objetos geométricos (puntos, líneas, polígonos) de los que se almacena sus atributos matemáticos. Permiten modificar el tamaño de una imagen sin sufrir la pérdida de calidad.

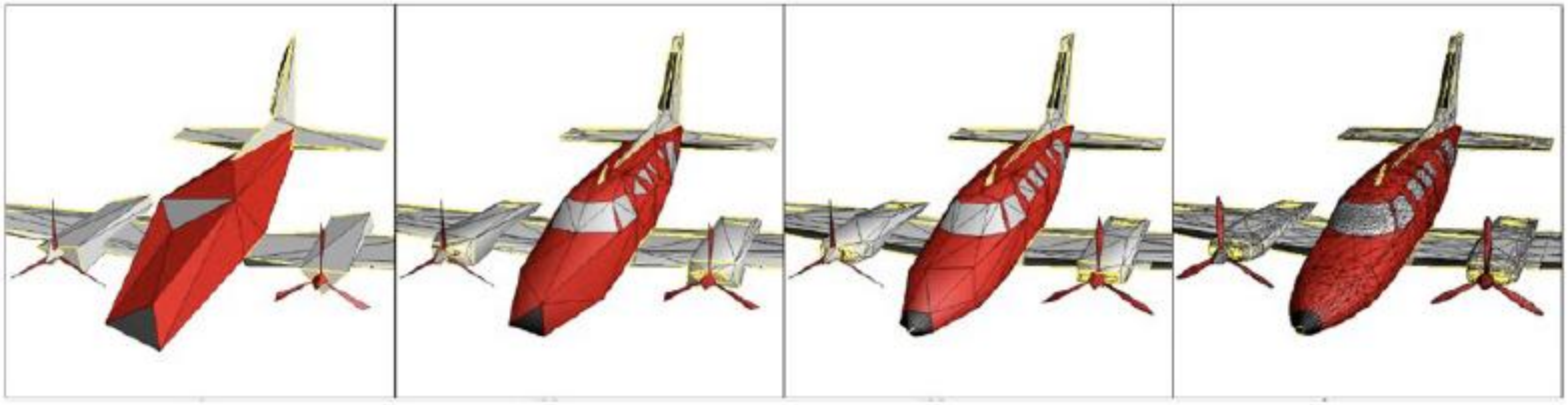
- **Renderizado**: Proceso de creación de una imagen a partir de un modelo. En esencia, consiste en calcular el color de cada uno de los pixeles de la imagen, considerando las propiedades del modelo a representar.
- **RayTracing** (trazado de rayos): técnica de renderizado que calcula el color de los píxeles simulando las trayectorias de los rayos de luz.



- El RayTracing produce imágenes de enorme calidad, pero requieren un gran esfuerzo de cómputo por lo que no puede utilizarse en aplicaciones en tiempo real.
- Es la técnica utilizada comúnmente en los efectos visuales de las películas de cine.
- (https://www.youtube.com/watch?v=wxSPdZl5sYw)
- (https://www.youtube.com/watch?v=gQ8noORAJuc)

- Una de las principales herramientas software basadas en RayTracing es ***Arnold,*** creada por un programador español (Marcos Fajardo) fundador de la empresa SolidAngle.
- SolidAngle fue absorbida por Autodesk. Actualmente tiene sede en Madrid y Londres.
- Arnold está incluido como módulo de renderizado de RayTracing en las herramientas Maya, Cinema4D, Houdini, 3DSMax y Katana.
- (https://www.arnoldrenderer.com/)

- **Rasterisation** (rasterizado): Técnica de renderizado que se basa en la proyección de figuras geométricas (gráficos vectoriales) sobre la matriz de píxeles (raster).
- **Vertex** (vértice): Un punto de un espacio en 2 o 3 dimensiones. Los vertex tienen asociadas propiedades como color, transparencia, dirección normal. Son el elemento básico para definir el modelo que se pretende representar. Son la base de los gráficos vectoriales.



- **Tesellation** (teselado): proceso de división de una superficie plana en un conjunto de triángulos o cuadriláteros.
- **Shader** (sombreador): programa que se ejecuta directamente en la GPU (procesador de la tarjeta gráfica) para realizar el renderizado. Se pueden programar diferentes etapas del proceso, lo que da lugar a diferentes shaders (vertex shader, fragment shader, geometry shader).
- **Viewport**: área de la ventana en la que se muestra el gráfico renderizado. Sus dimensiones determinan las coordenadas físicas (pixeles) a las que hay que transformar la imagen.

- **Clipping volume**: volumen del espacio 3D que se utiliza para calcular la imagen renderizada. Lo que está fuera del clipping volume queda fuera de la imagen generada
- **Aliasing**: efecto visual que permite apreciar que una imagen está compuesta de pixels. Para evitarlo existen técnicas de antialiasing, que suavizan la imagen para evitar este efecto
- **Pipeline**: conjunto de etapas en las que se divide el proceso de renderizado.

2.1 Informática gráfica

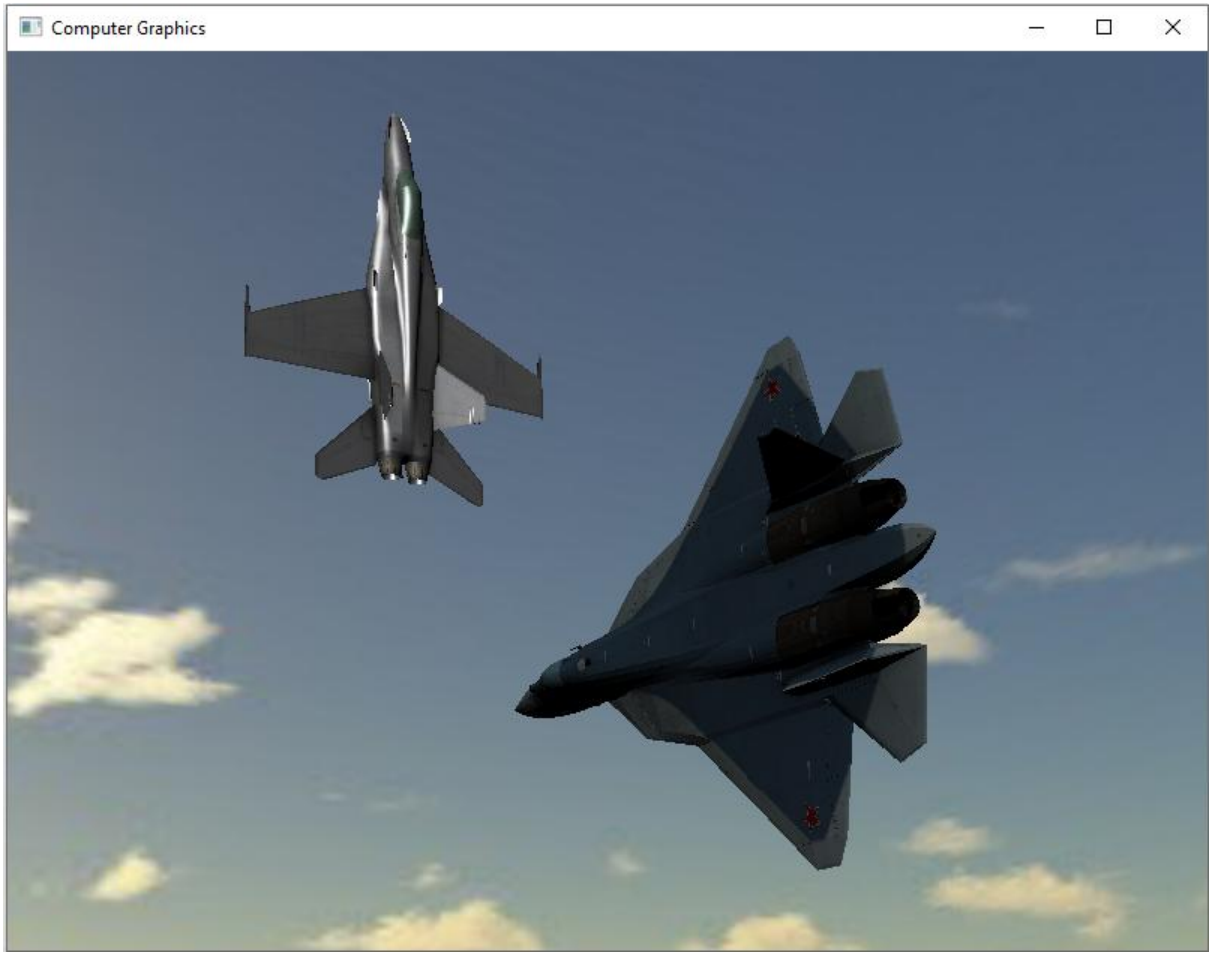
2.2 Principios básicos de representación en 3D

2.3 Introducción a OpenGL

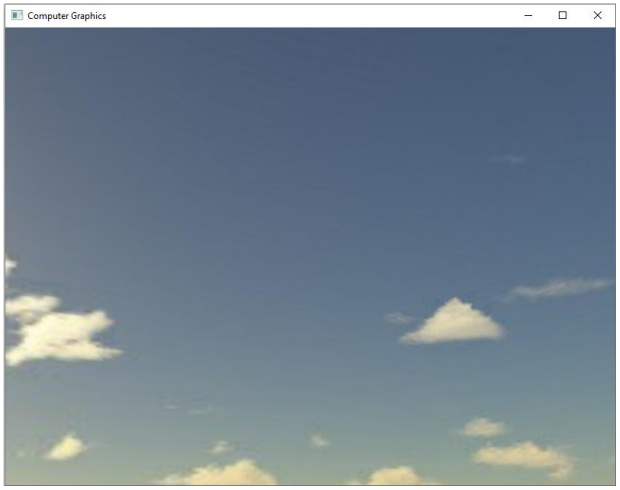
2.4 La evolución de OpenGL

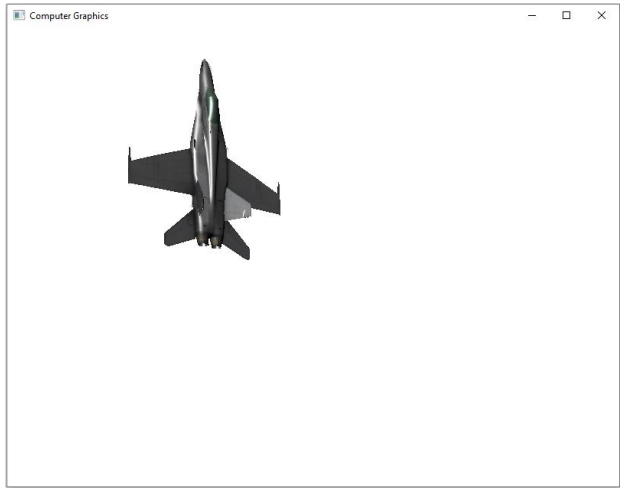
2.5 Aspectos básicos de OpenGL

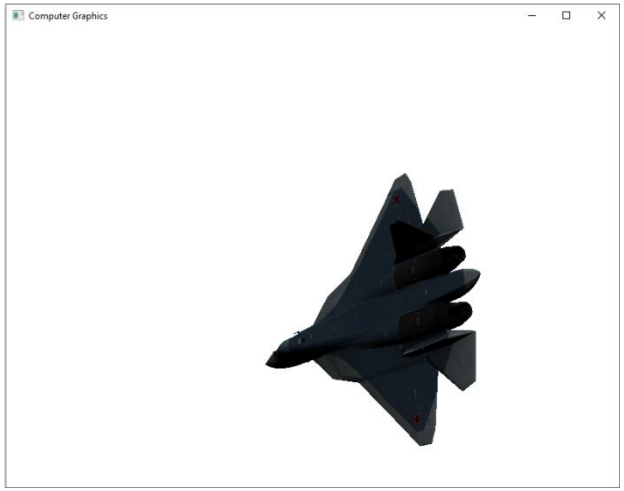
- ¿Cómo se genera una imagen?



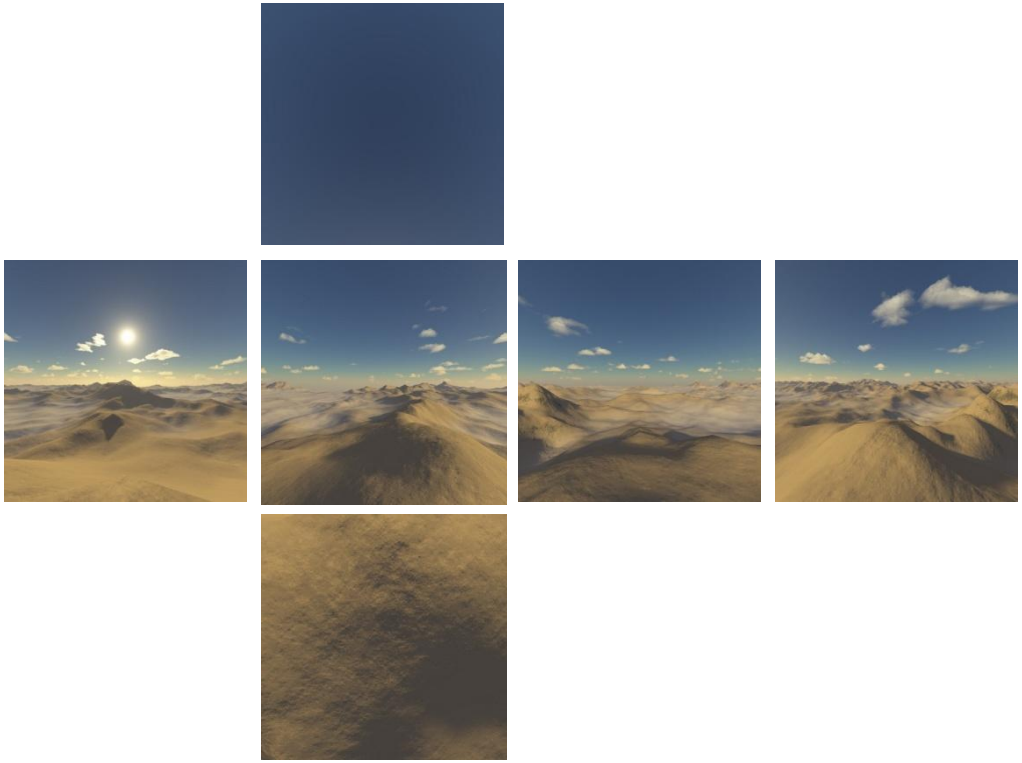
- El esquema básico para generar un imagen consiste en dibujar el fondo de la imagen y, a continuación, dibujar los objeto que forman parte de la escena.



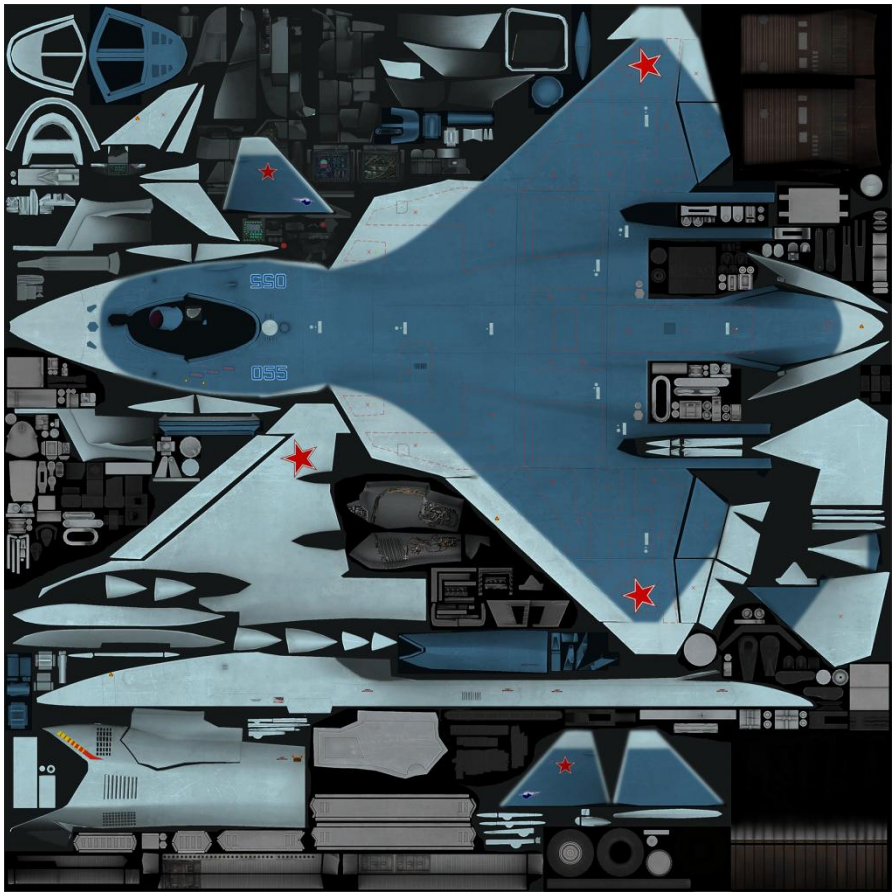




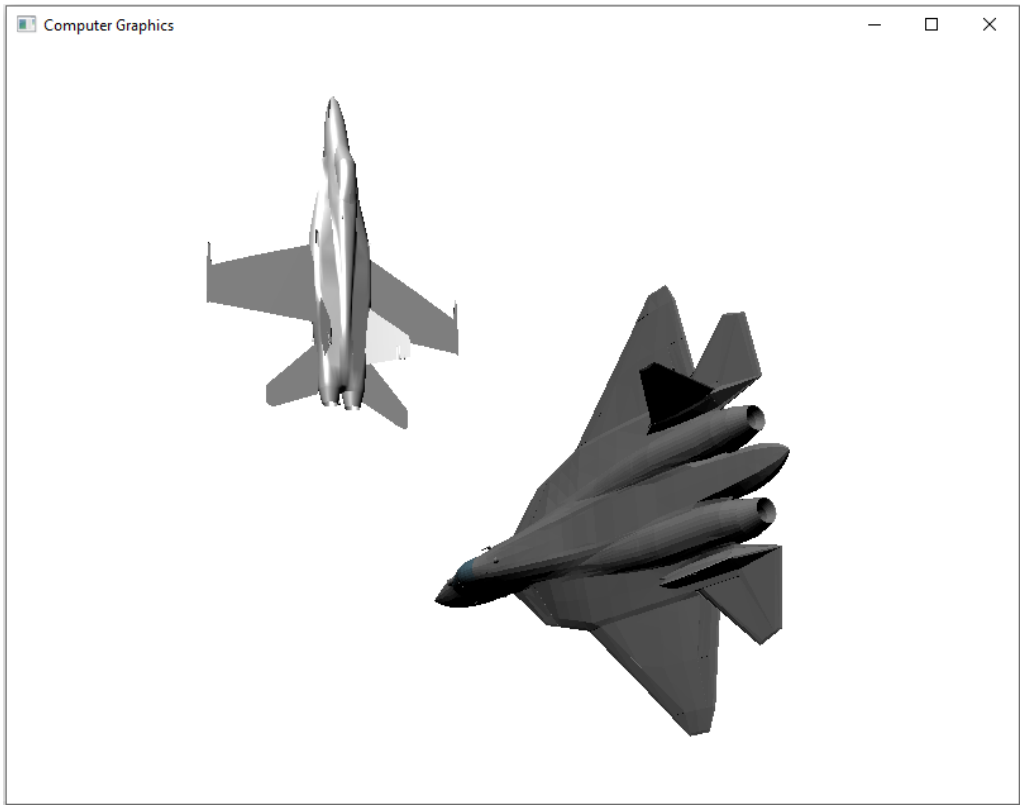
- El fondo se genera a partir de un conjunto de seis imágenes en forma de cubo (lo que se conoce como *cubemap*). Estas imágenes se proyectan sobre las caras de un cubo de forma que el observador está colocado en el centro de ese cubo (*skybox*).



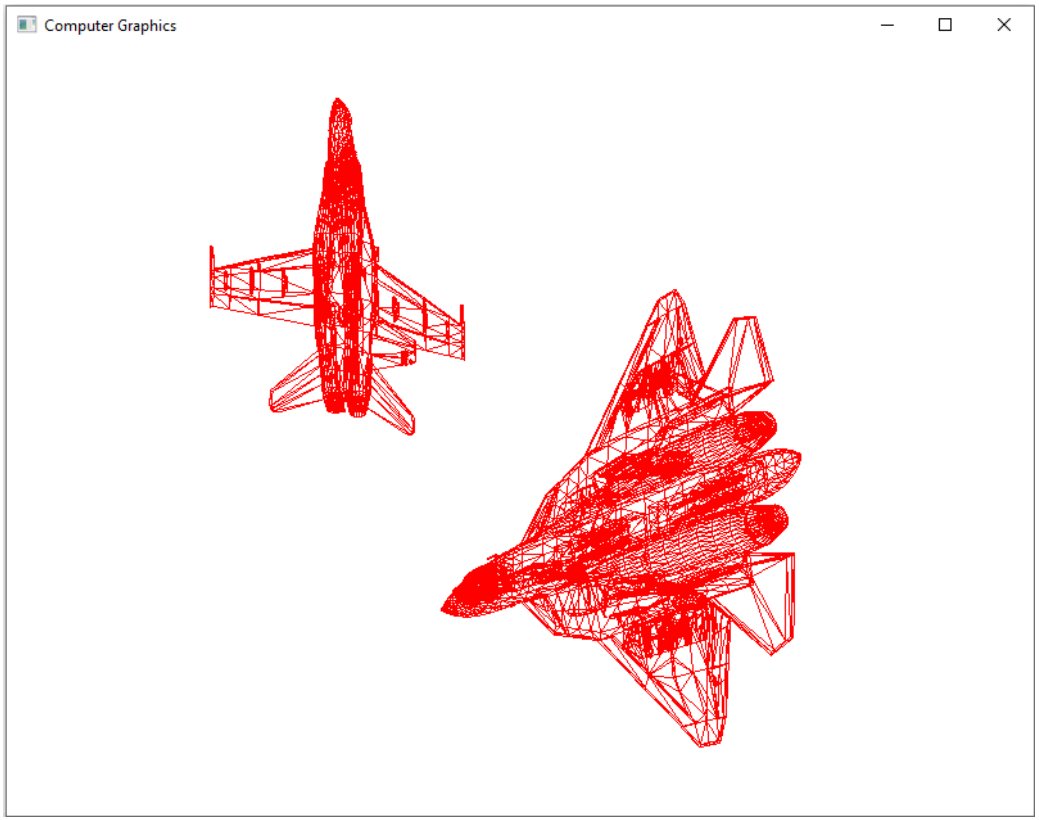
- Para dibujar los objetos se utilizan texturas, que describen el aspecto que tiene la superficie del objeto.



- Si quitamos la textura a los objetos podemos ver como la superficie se dibuja más clara o más oscura en base a la iluminación.



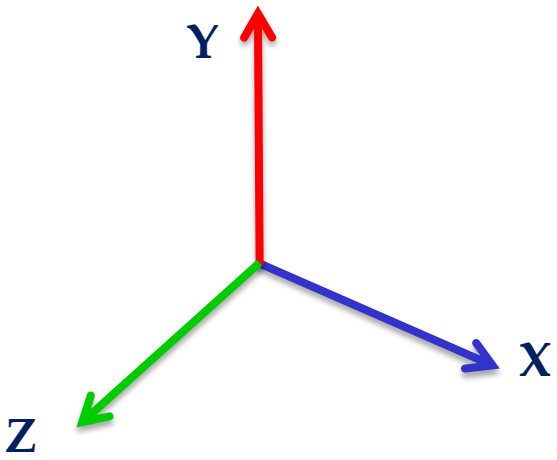
- La superficie de los objetos se basa en dibujar una malla (*mesh*) formada por vértices unidos por líneas. Cada superficie triangular se conoce como una primitiva.



- La definición de un objeto está formada por la descripción de un conjunto de vértices (de los que se indican su posición, su normal y su coordenada de textura) y una lista de caras formadas por la referencia a tres vértices.

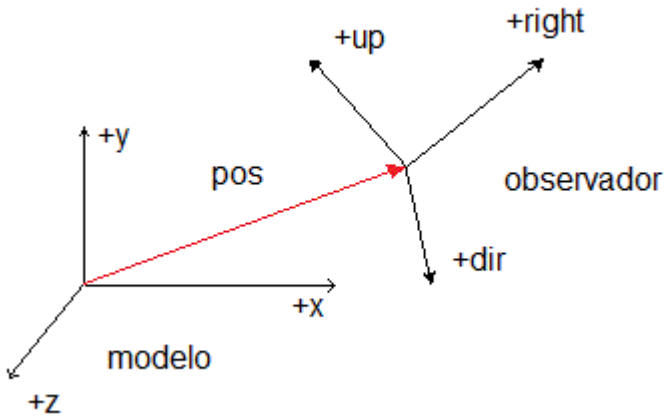
```
p_vertices_0[438][3] = {  { -0.521909f, 2.569959f, -5.647568f },  …. };

p_textures_0[438][2] = {  { 0.176717f, 0.662548f },  … };

p_normals_0[438][3] = {  { -0.913319f, 0.375991f, -0.156459f },  …};

p_indexes_0[146][3] = {  { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 },  …};
```

- Sistema de coordenadas:
  - Para situar los objetos en el espacio 3D es necesario trabajar sobre un sistema de coordenadas X-Y-Z. La dirección de los ejes se rige por la "regla de la mano derecha".



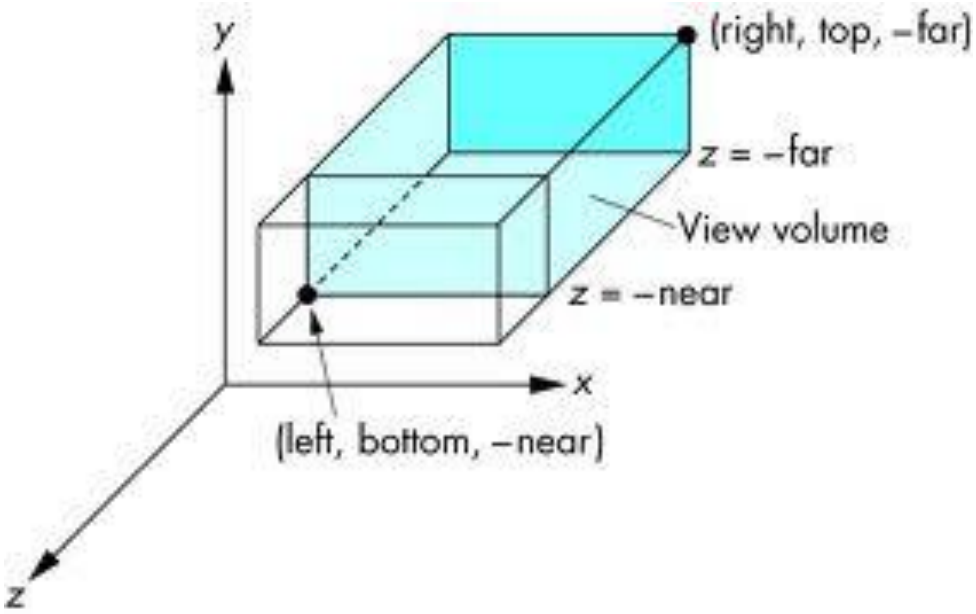
- Sistemas de coordenadas
  - A la hora de generar modelos en 3D se utilizan diferentes sistemas de coordenadas.
  - El sistema de coordenadas local de un objeto considera el origen de coordenadas en un punto fijo del objeto (independientemente de su posición u orientación).
  - El sistema de coordenadas del modelo permite establecer la posición y orientación de los objetos presentes en el modelo.
  - El sistema de coordenadas del observador considera el origen de coordenadas en la posición del observador.

- Transformaciones de coordenadas
  - Para realizar transformaciones de coordenadas se utilizan matrices de transformación.

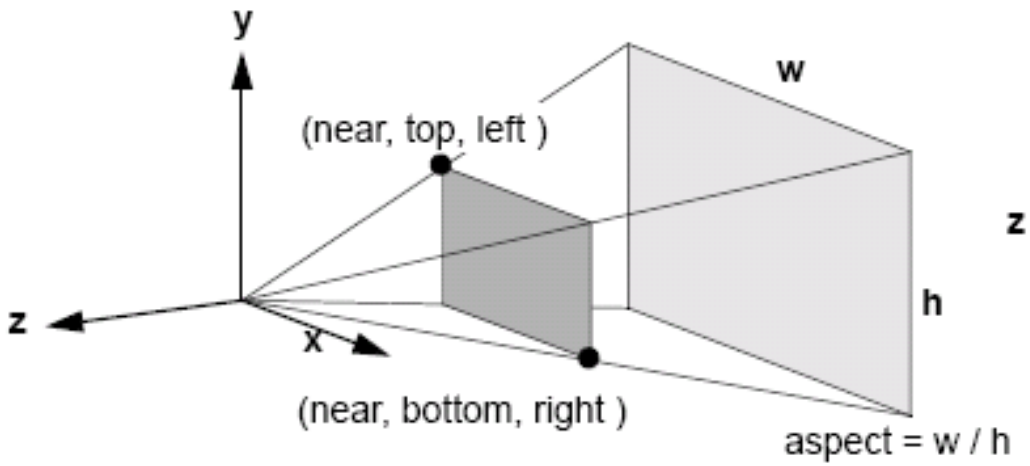


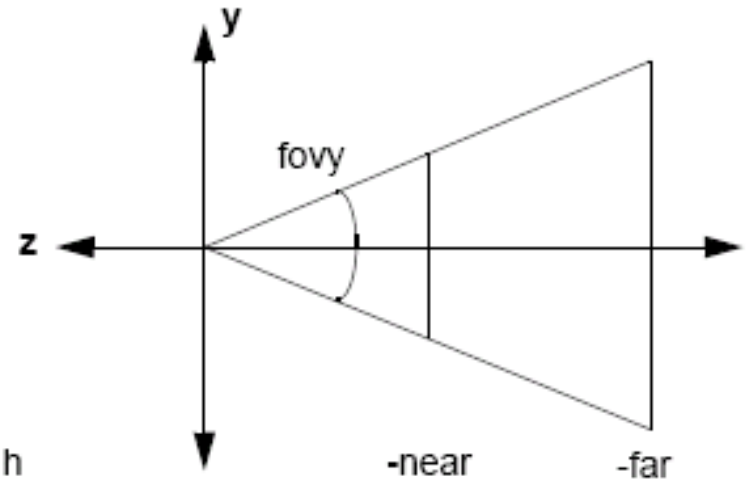
- Proyecciones:
  - Para dibujar la imagen, es necesario proyectar sobre un plano los polígonos que forman el modelo.
  - Hay dos tipos de proyección: ortográfica y en perspectiva.

- Proyección ortográfica:
  - Es una proyección paralela al eje Z.
  - El volumen a proyectar (clipping volume) es un prisma rectangular.
  - La proyección consiste en eliminar la componente Z.

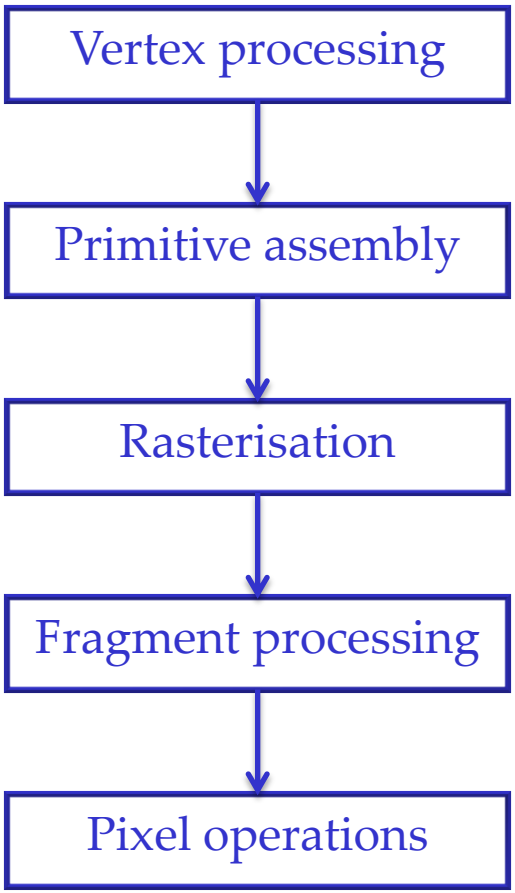


- Proyecciones en perspectiva:
  - Es una proyección en la que los objetos más cercanos aparecen más grandes que los objetos más elejados.
  - El volumen a proyectar (*clipping volume*) es un tronco de pirámide (*frustum*).

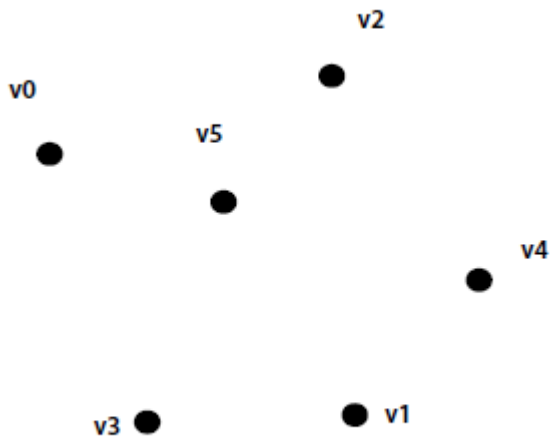




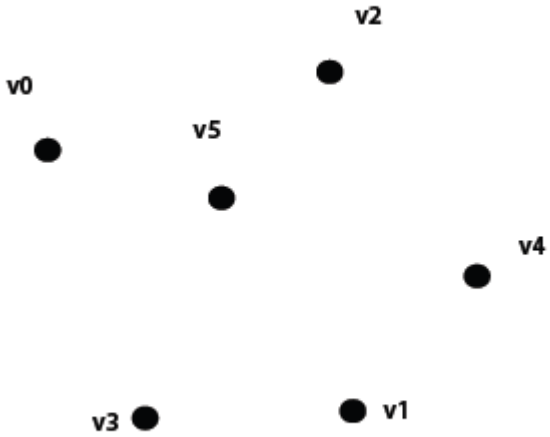
- Etapas del proceso básico de renderizado (*pipeline*):

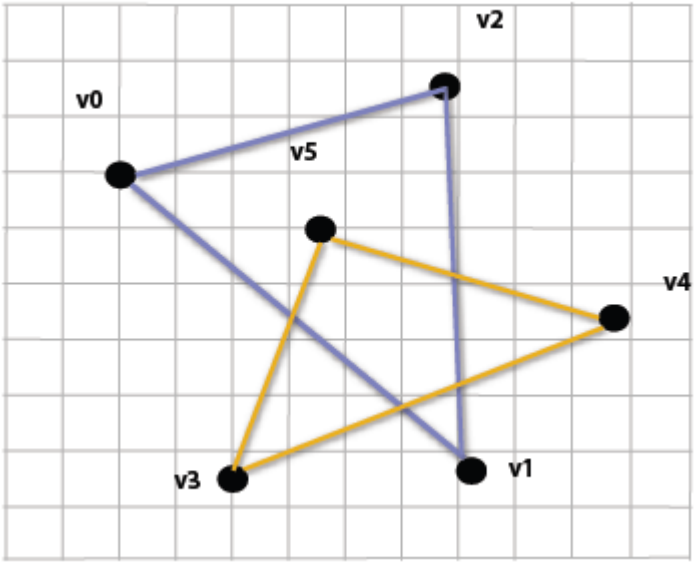


- Vertex processing:
  - Los vértices son proyectados sobre el área de dibujo (clipping area)

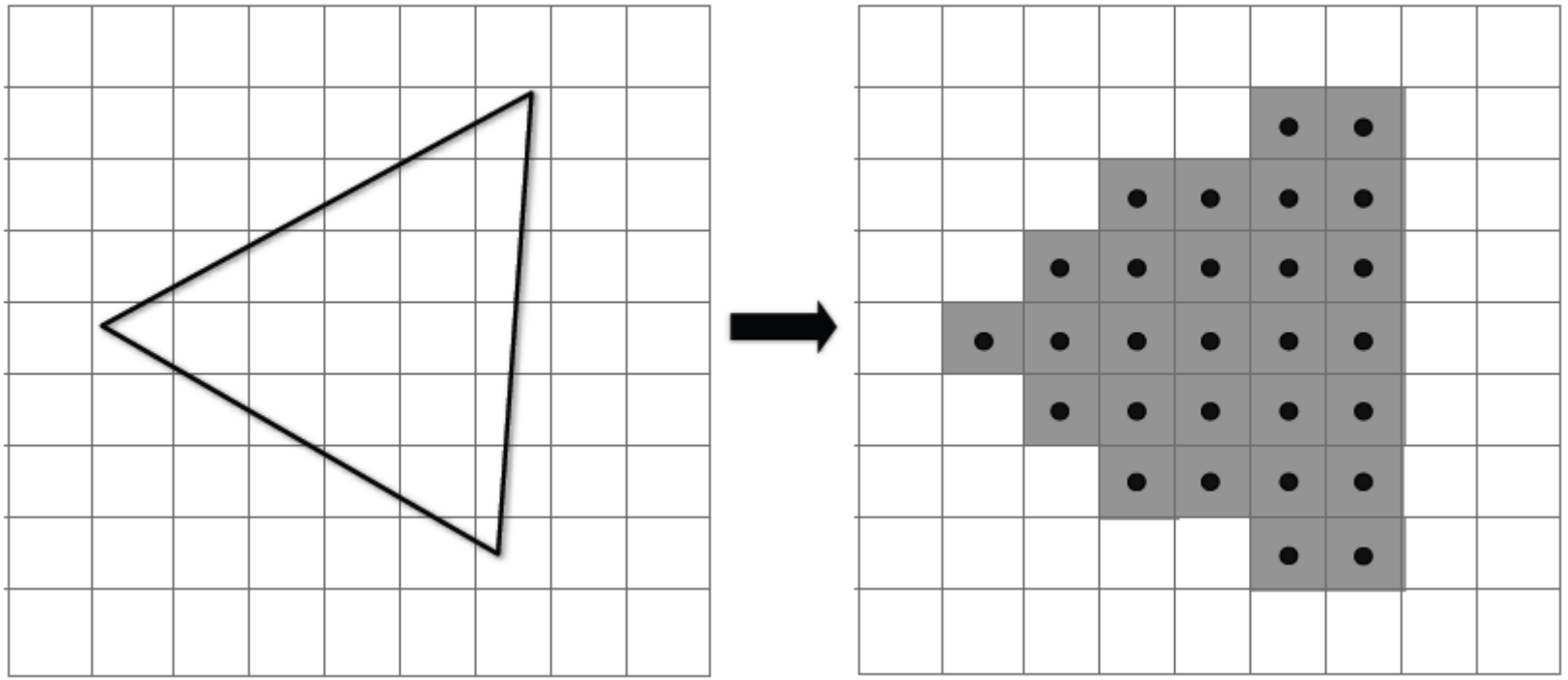


- Primitive processing:
  - Los vértices se organizan en primitivas (triángulos) y se detecta si están ocultos o fuera del área de dibujo

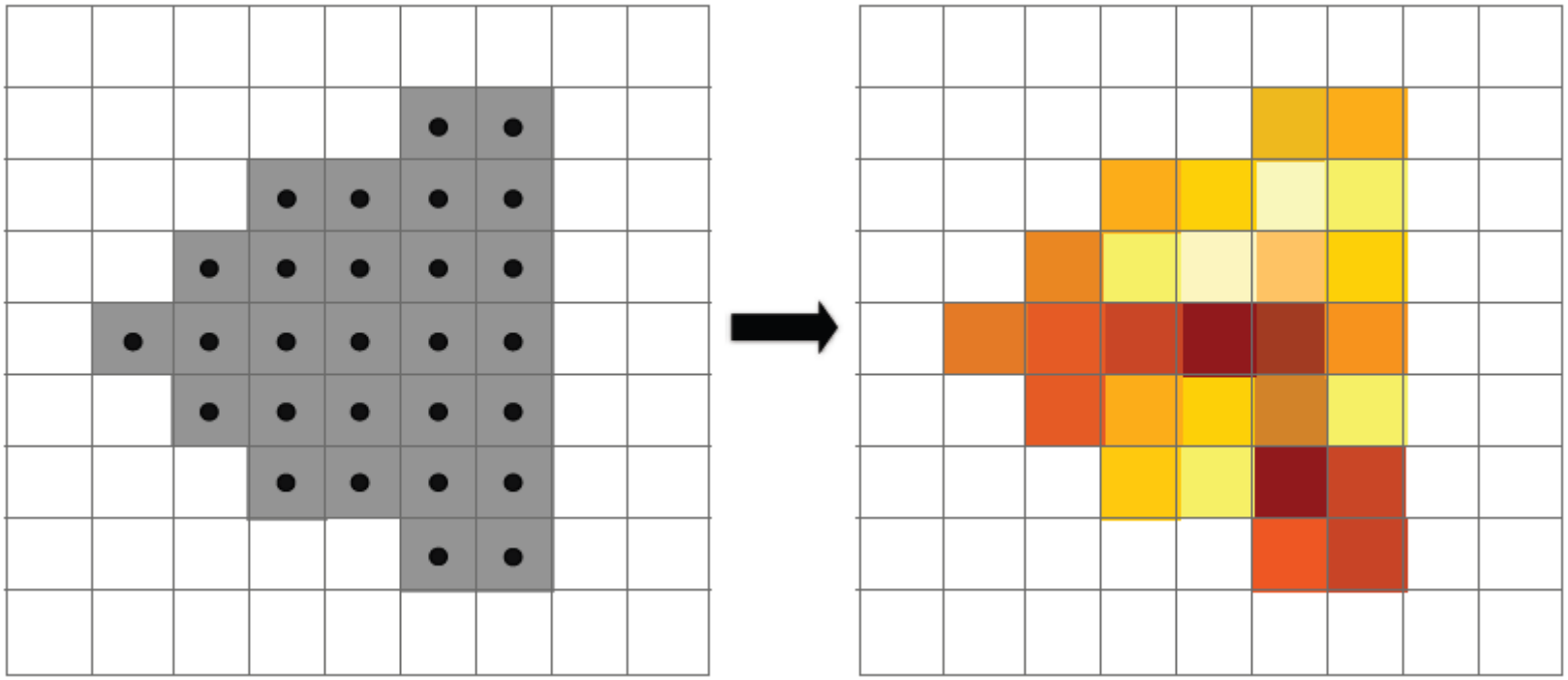




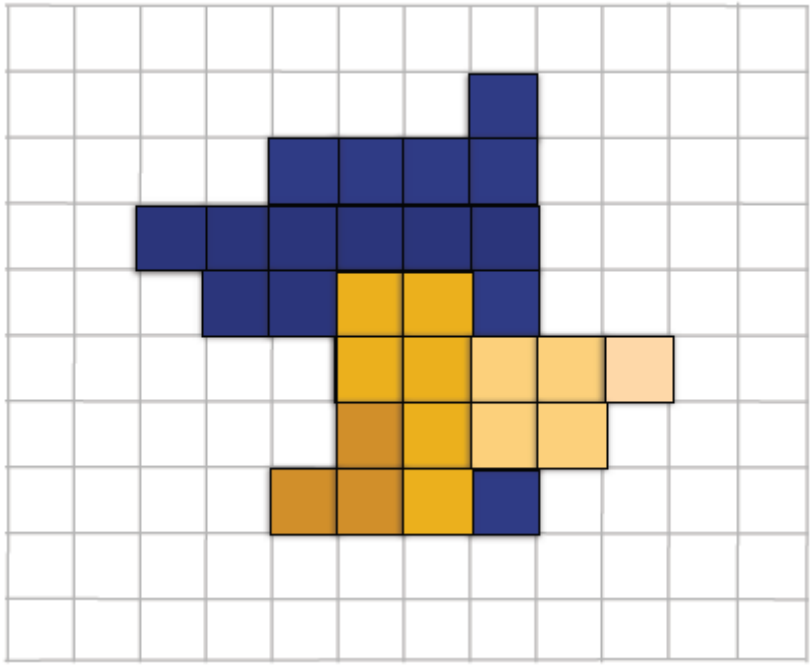
- Rasterisation:
  - Las primitivas se transforman en “pixel fragments”



- Fragment processing:
  - Se calcula el color de cada pixel del fragmento utilizando técnicas de coloreado, iluminado, sombreado y textura.



- Pixel operations:
  - Se mezclan los diferentes fragmentos para determinar el valor real de cada pixel. Se utiliza información de la profundidad de cada pixel y su nivel de transparencia.



2.1 Informática gráfica

2.2 Principios básicos de representación en 3D

2.3 Introducción a OpenGL

2.4 La evolución de OpenGL

2.5 Aspectos básicos de OpenGL

- En 1981, el Dr. James H. Clark (entonces profesor asociado de Ingeniería Eléctrica en la Universidad de Stanford) diseñó el primer hardware dedicado a la generación de imágenes basadas en modelos geométricos (***Geometry Engine***).
- En 1982, Jim Clark abandonó la universidad para formar la compañía Silicon Graphics Inc. junto con Abbey Silverstone.
- Dos meses más tarde se unió un grupo de estudiantes graduados de Stanford incluyendo a Kurt Akeley, Tom Davis, Rocky Rhodes, Mark Hannah, Herb Kuta, y Mark Grossman.

●SGI 創始者 Jim Clark 氏

- En 1984, SGI sacó al mercado su primer producto: IRIS 1000 (IRIS son siglas de Integrated Raster Imaging System, “sistema integrado de la proyección de imagen ")
- Se trataba de terminales gráficas diseñadas para ser conectadas a una computadora DEC VAX, manejando solamente la pantalla.
- Estaban basadas en el microprocesador Motorola 68000.

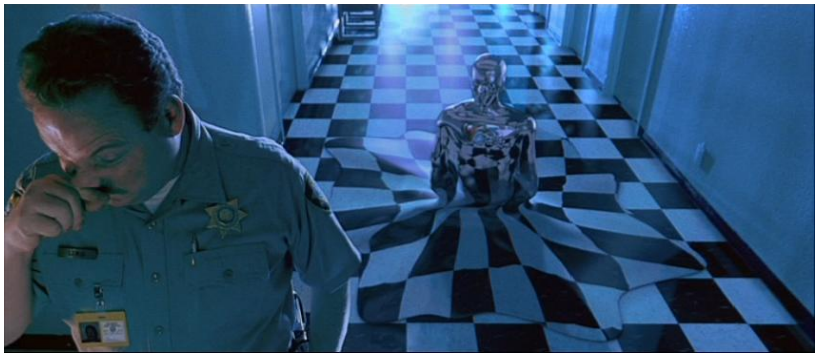
●SGI 初のハードウエア「IRIS 1000」グラフィックスターミナル

- SGI mejoró enseguida sus máquinas hasta convertirlas en estaciones de trabajo.
- En 1985 salió al mercado la serie IRIS 2000, un conjunto de estaciones de trabajo basadas en el sistema operativo UNIX System V, procesador Motorola 68010, discos duros, unidades de cinta y placas Ethernet.
- El sistema incluía hardware dedicado a la generación de gráficos (Geometry Engine) y software preparado para aprovechar las características de este motor gráfico (IRIS Graphics Library)

(http://www.youtube.com/watch?v=9EEY87HAHzk)

- En 1987 aparece la serie IRIS 3000, basada en el procesador Motorola 68020.
- En 1989 SGI comenzó a utilizar procesadores basados en la arquitectura MIPS. La versión de Unix utilizada en las estaciones de trabajo de SGI se bautizó como IRIX. La aceleración gráfica se basaba en hardware denominado Onyx Reality Engine.

- Durante la década de 1990 los sistemas y estaciones de trabajo de Silicon Graphics fueron el soporte de la mayor parte de las aplicaciones gráficas desarrolladas en esa época.
- Las primeras películas con gráficos realistas fueron The Abyss (1989) y Terminator II (1991). Los efectos 3D se encargaron a IL&M, que utilizó el software Alias (de Alias Research) sobre estaciones de trabajo SGI 4D/70G y SGI 4D/80GT.



- Hasta la segunda generación de las máquinas Onyx Reality Engine, SGI ofreció acceso a sus subsistemas gráficos 3D de alto rendimiento con un API propietario conocido como IrisGL. Mientras que se agregaron más características al pasar los años, IrisGL se volvió más complicado de mantener e incómodo de utilizar.
- En 1992, SGI se decidió a limpiar y reformar IrisGL e inició un movimiento para permitir que el API resultante, llamado OpenGL, fuera licenciado a un precio económico para los competidores de SGI.

- Para desarrollar un estándar abierto se creó un consorcio industrial llamado OpenGL Architecture Review Board (OpenGL ARB), formado inicialmente por SGI, DEC, IBM, Intel y Microsoft.
- La primera versión de OpenGL se publicó el 1 de julio de 1992.

- En 1995, Microsoft decidió crear su propia librería gráfica para integrarla en su nuevo sistema operativo Windows 95. Para ello compró la compañía RenderMorphics (creada por Servan Keondjian en 1992), que había desarrollado la librería gráfica Reality Lab. El resultado fue la librería gráfica Direct3D.

- La primera versión de Direct3D resultó muy difícil de programar para los fabricantes de tarjetas gráficas y desarrolladores de juegos, por lo que solicitaron a Microsoft que facilitara la integración de OpenGL en su sistema. Microsoft distribuyó a los fabricantes un driver kit que facilitaba el desarrollo de drivers de OpenGL para Windows NT y Windows 98.
- Justo antes de la publicación de Windows 98, Microsoft anunció que no extendía la licencia de uso del driver kit y prohibía a los fabricantes distribuir los drivers desarrollados a partir del kit.
- Al mismo tiempo, SGI anunciaba su intención de crear estaciones de trabajo basadas en WindowsNT y un acuerdo con Microsoft para desarrollar una nueva librería, denominada Farenheit, que sustituiría a OpenGL.

- Sin embargo, los fabricantes reaccionaron desarrollando drivers de OpenGL directamente a partir de la especificación del estándar y los desarrolladores continuaron utilizando masivamente OpenGL frente a Direct3D.
- En los siguientes años, Microsoft abandonó el OpenGL ARB y se centró en mejorar las características de Direct3D.
- En 2006, la quiebra de SGI hizo que el desarrollo de OpenGL pasara a manos de The Khronos Group, un consorcio de empresas dedicado al desarrollo de estándares abiertos.
- El Khronos Group se organiza en subgrupos. El dedicado a OpenGL se denomina OpenGL ARM Working Group.

- Cuando un fabricante quiere distribuir un producto compatible con OpenGL, debe solicitar una licencia al Khronos Group. Para obtener la licencia OpenGL el consorcio realiza pruebas para verificar que el producto cumple con las especificaciones del estándar.

- Miembros promotores del Khronos Group

- Miembros contribuyentes del Khronos Group

- Miembros asociados del Khronos Group

- Miembros académicos del Khronos Group

2.1 Informática gráfica

2.2 Principios básicos de representación en 3D

2.3 Introducción a OpenGL

2.4 La evolución de OpenGL

2.5 Aspectos básicos de OpenGL

- La primera versión de OpenGL se publicó en 1992. Desde entonces se han producido numerosas revisiones que han ido ampliando las funcionalidades de la librería en paralelo a las mejoras introducidas en las tarjetas gráficas.
- OpenGL 1.0 (julio de 1992)
- OpenGL 1.1 (enero de 1997)
- OpenGL 1.2 (marzo de 1998)
- OpenGL 1.2.1 (octubre de 1998)
- OpenGL 1.3 (agosto de 2001)
- OpenGL 1.4 (agosto de 2001)
- OpenGL 1.5 (julio de 2003)

- En el año 2001 aparecen las primeras tarjetas gráficas en las que se incluye la posibilidad de programar externamente algunas etapas del pipeline (NVIDIA GeForce 3, ATI Raedon 8500).
- Las etapas programables eran *vertex_program*, que se ejecutaba sobre cada vértice, y *fragment_program*, que se ejecutaba sobre cada pixel de cada fragmento.
- Inicialmente, esta programación debía hacerse en un ensamblador específico y con un tamaño de programa reducido. Pese a las limitaciones, estas etapas programables permitían incorporar nuevas técnicas de interpolación, mezcla o sombreado sin necesidad de añadirlas como opciones del estándar OpenGL.

- OpenGL 2.0 (septiembre de 2004)
  - A partir de 2003, las tarjetas gráficas mejoraron notablemente las capacidades de las etapas programables (NVIDIA GeForce FX, ATI Radeon 9700, 3DLab WildCat VP)
  - A partir de esta versión, OpenGL introdujo un lenguaje de programación de shaders denominado GLSL.
  - Microsoft desarrolla un lenguaje similar para la programación de shaders denominado HLSL, utilizado en su biblioteca Direct3D.

- OpenGL 3.0 (agosto de 2008)
  - Esta versión introdujo un mecanismo para declarar obsoletas (deprecate) algunas funciones de la librería.
- OpenGL 3.1 (marzo de 2009)
  - Las funciones obsoletas fueron eliminadas.
- OpenGL 3.2 (agosto de 2009)
  - La compañía NVIDIA se negó a eliminar las características declaradas obsoletas y aseguró que sus versiones de OpenGL siempre serían compatibles hacia atrás.
  - En esta revisión se decidió dividir el estándar en dos versiones: *Core profile* (incluye solo las funciones activas) y *Compatibility profile* (incluye las funcones obsoletas).

- OpenGL 4.0 (marzo de 2010)
- OpenGL 4.1 (julio de 2010)
- OpenGL 4.2 (agosto de 2011)
- OpenGL 4.3 (agosto de 2012)
- OpenGL 4.4 (julio de 2013)
- OpenGL 4.5 (agosto de 2014)
- OpenGL 4.6 (2017)

- OpenGL sufre problemas debidos a su diseño original de 1992
  - OpenGL API es una máquina de estados.
  - El estado de OpenGL está asociado a un único contexto gráfico.
  - OpenGL esconde lo que la hace realmente la GPU.
  - Alto consumo de CPU: comprobación de errores en cada llamada, compilación de shaders se retrasa hasta cuando se ejecuta glDraw*(), etc.
  - Cada implementación de OpenGL incluye un compilador propio de GLSL

- En 2016 el Khronos Group presentó la primera versión de su nueva biblioteca gráfica que denominó Vulkan.
  - Vulkan 1.0 (febrero 2016)
  - Vulkan 1.1 (marzo 2018)
  - Vulkan 1.2 (enero 2020)
  - Vulkan 1.3 (febrero 2022)

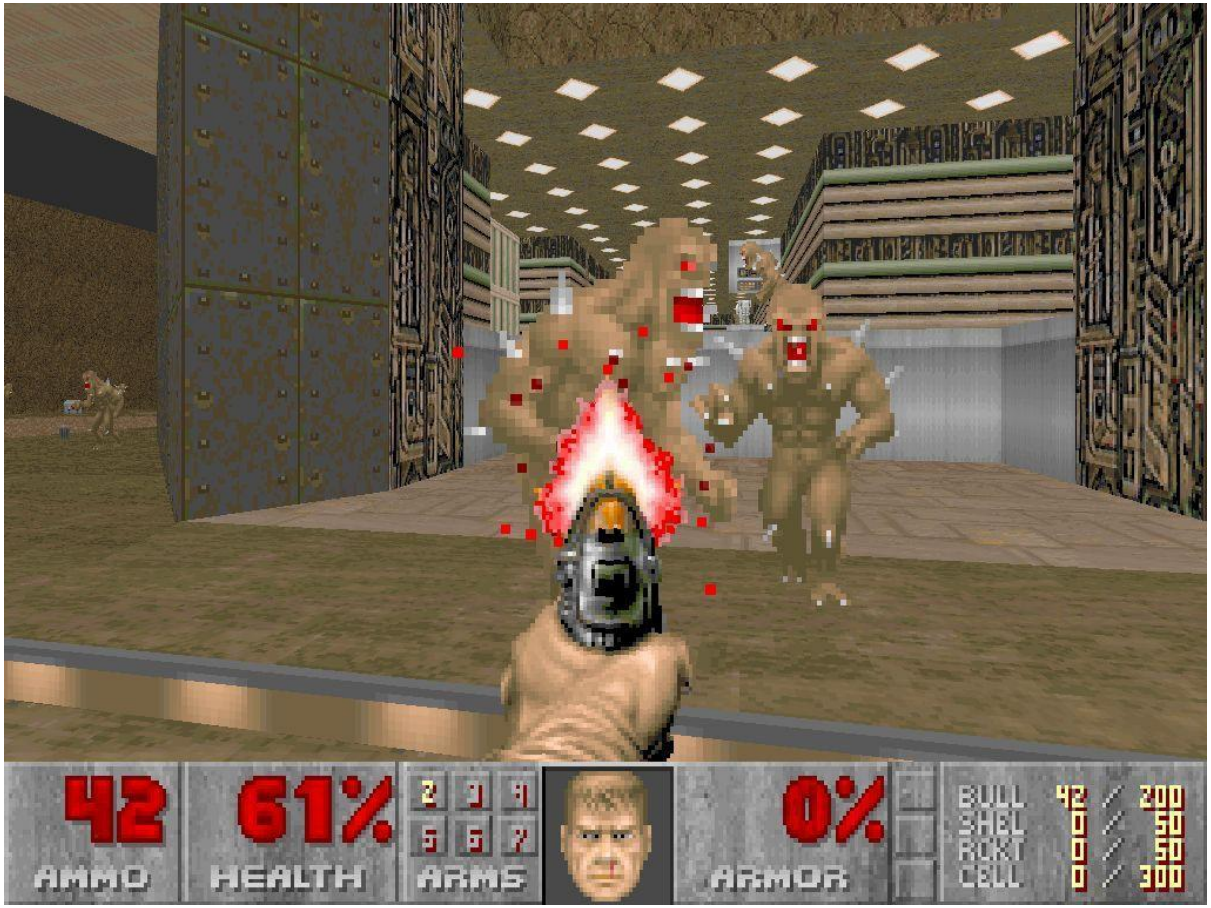
- La tecnología de GPUs ha avanzado muchísimo

VGA-EISA (1991)

Nvidia GeForce RTX 4090

Alone in the Dark (1992) Infogrames

Doom II (1994) Id Software



Doom (2016) Bethesda

2.1 Informática gráfica

2.2 Principios básicos de representación en 3D

2.3 Introducción a OpenGL

2.4 La evolución de OpenGL

2.5 Aspectos básicos de OpenGL

- OpenGL es una biblioteca de funciones escritas en C.
- Estas funciones describen un conjunto de comandos que permiten crear las imágenes.
- Las funciones permiten acceder a la memoria gráfica (para almacenar los atributos de los vértices o las texturas), configurar las etapas de renderizado programables (shaders), configurar las etapas de renderizado no programables (activando y configurando opciones), lanzar primitivas, etc.
- OpenGL no incluye funciones de manejo de ventanas, ni de interfaz de usuario (eventos de ratón o teclado), ni de flujos de entrada o salida. Cada entorno (Mac, Linux, MS-Windows, …) tiene su propia forma de adaptar OpenGL a su interfaz gráfica.

- Hay dos tipos de implementaciones de OpenGL: implementaciones genéricas e implementaciones hardware.
- Las implementaciones genéricas son implementaciones software del conjunto de funciones contenido en OpenGL. Estas funciones se apoyan en la interfaz gráfica de la plataforma correspondiente y no utilizan ninguna característica de aceleración gráfica. Son implementaciones muy lentas, basadas en las primeras versiones de OpenGL.
- Wiggle y Mesa3D son ejemplos de implementaciones genéricas.
- Las implementaciones hardware son desarrolladas por los fabricantes de tarjetas gráficas y distribuidas como drivers de estas tarjetas.

- En las versiones antiguas de OpenGL (1.0 a 2.0) las funciones de la librería se ejecutaban en la CPU y generaban comandos que se almacenaban en un buffer. Posteriormente este buffer se enviabba a la GPU (tarjeta gráfica) y se ejecutaba.
- En las versiones modernas de OpenGL (a partir de la versión 3.0) se hace un uso más intensivo de la memoria interna de la tarjeta gráfica. Las funciones se dedican fundamentalmente a almacenar la información en esa memoria (atributos de los vértices, texturas, programas de los shaders, …). Los comandos se utilizan para configurar las etapas no programables y lanzar primitivas que ponen en marcha el pipeline de renderizado.

- OpenGL define sus propios tipos de datos para facilitar la portabilidad entre distintas plataformas. Los tipos son:

OpenGL	Representación interna	Definición en C
GLbyte	entero de 8 bits	signed char
GLshort	entero de 16 bits	short
GLint, GLsizei	entero de 32 bits	long
GLfloat	flotante de 32 bits	float
GLclampf	puntero	
GLdouble	flotante de 64 bits	double
GLclampd	puntero	
GLubyte	entero sin signo de 8 bits	unsigned char
GLboolean	entero	
GLushort	entero sin signo de 16 bits	unsigned short
GLuint, GLenum	entero sin signo de 32 bits	unsigned long
GLbitfield	entero	
GLchar	carácter de 8 bits	char
GLsizeiptr, GLintptr	puntero nativo	ptrdiff_t

- La mayoría de las funciones de OpenGL tienen distintas versiones en función del tipo de dato que utilizan.
- El esquema general de los nombres de las funciones es:

<prefijo> <comando><numero de argumentos><tipo de argumentos>

- Por ejemplo:

glColor3f(GLfloat r, GLfloat g, GLfloat b)

- El prefijo indica la biblioteca a la que pertenece la función. Todas las funciones de OpenGL tienen el prefijo “gl”.
- Existen otras librerías auxiliares de OpenGL, cuyas funciones utilizan prefijos diferentes. Por ejemplo, la librería GLU (OpenGL utility library) utiliza el prefijo “glu”. La librería GLUT (OpenGL utility toolkit) utiliza el prefijo “glut”.

- **GLEW: The OpenGL Extension Wrangler Library**
  - Debido al amplio número de versiones que existen de OpenGL, a menudo la programación gráfica puede presentar problemas.
  - Si compilamos en un equipo con una cierta versión e intentamos ejecutar el programa en otro equipo con una versión inferior se producirán errores.
  - La librería GLEW permite adaptar las versiones de OpenGL, comprobando en tiempo de ejecución las extensiones soportadas en el equipo de destino.