MSc Computer Games and Entertainment

Maths & Graphics Unit 2011/12

Lecturer: Gareth Edwards

PART 1:

Mathematics for Computer Graphics

Mathematics for Computer Games

- This Course on Mathematics for Computer Games covers a wide range of topics; from elementary subjects such as Cartesian coordinates systems and subjects such as the dot and cross product, through vectors, matrices, transforms, parametric, curves, to simulation, global illumination model, programmable shaders, and real and non-real time rendering techniques.

- The objective of this course if to ensure that all Students have a broad understanding of the Mathematics required to create and render a 3D Scene.

- For those Students who are already familiar with much of the subject matter, this course might be considered as a refresher; for others it should be a “Call to Arms”; an opportunity to study and understand those parts of the subject matter covered of which they are unsure or unfamiliar.

Course Context – The 3D Pipeline

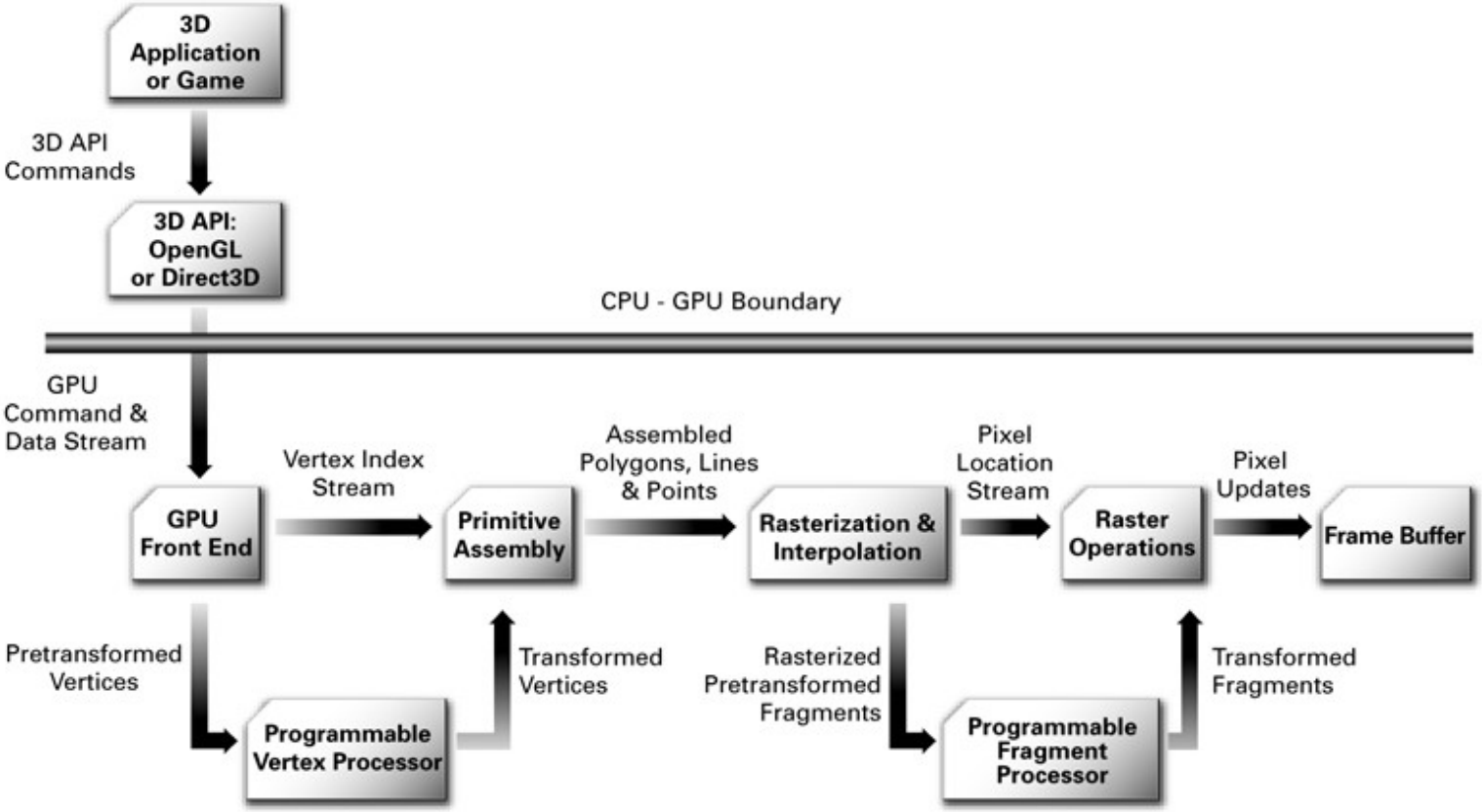
The pipeline is the “engine” that creates images from a description of a 3D scene.

There are 3 conceptual stages of the pipeline:

1. **Application** – executed on the CPU, including creating and then sending graphic primitives to be stored on specialised hardware - the GPU.
2. **Geometry** – from within an application perform “geometrical” operations, which are executed on the GPU, on the graphic primitives stored on the GPU.
3. **Rasterizer** – from within an application render a 3D scene on the GPU.

Cartesian coordinate plane with four points.

Course Context – The 3D Pipeline



Graphic Primitives

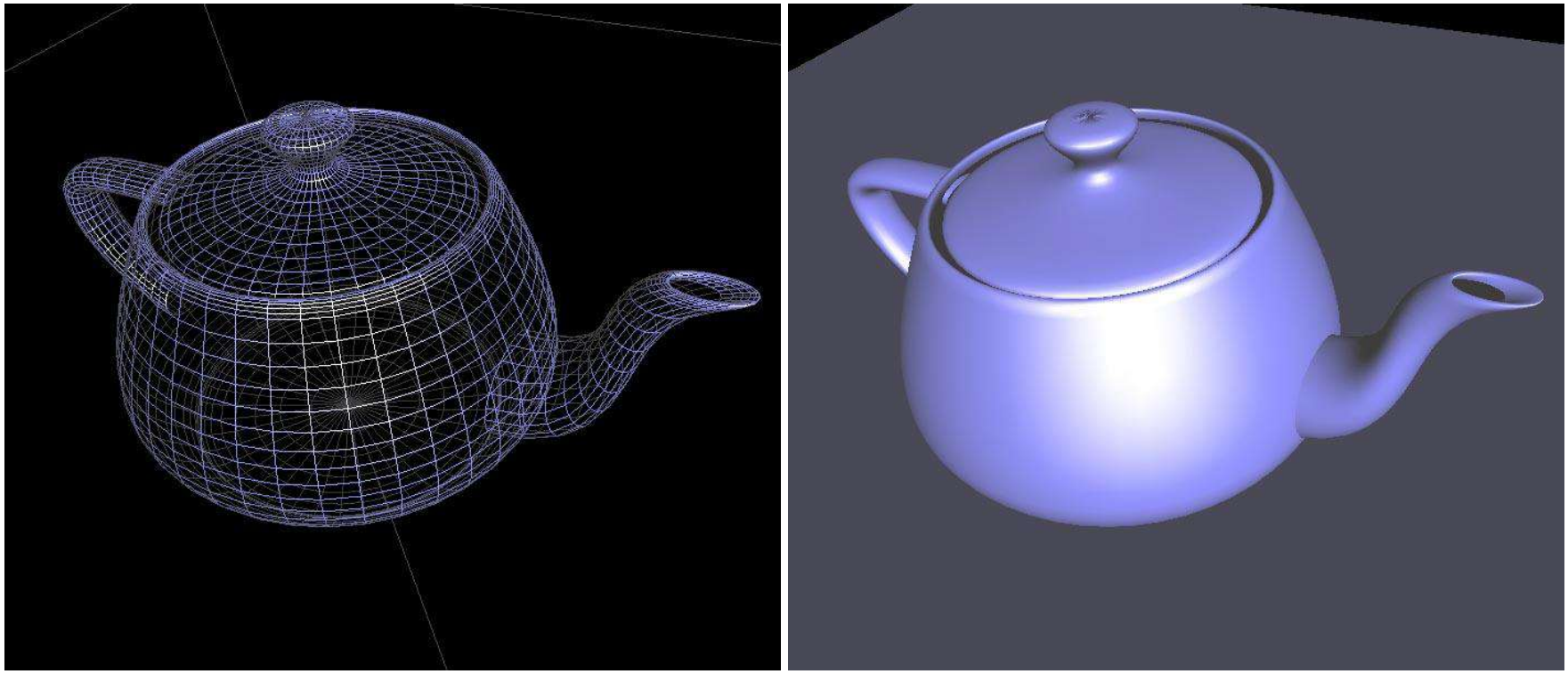
The 3D Pipeline uses the GPU to render 3D scenes in real time.

3D scenes are made up of objects.

These objects are constructed from geometry, where:

- Geometry is either pre-created using an interactive modeling package or created procedurally within an application

Graphic Primitives – the Utah Teapot



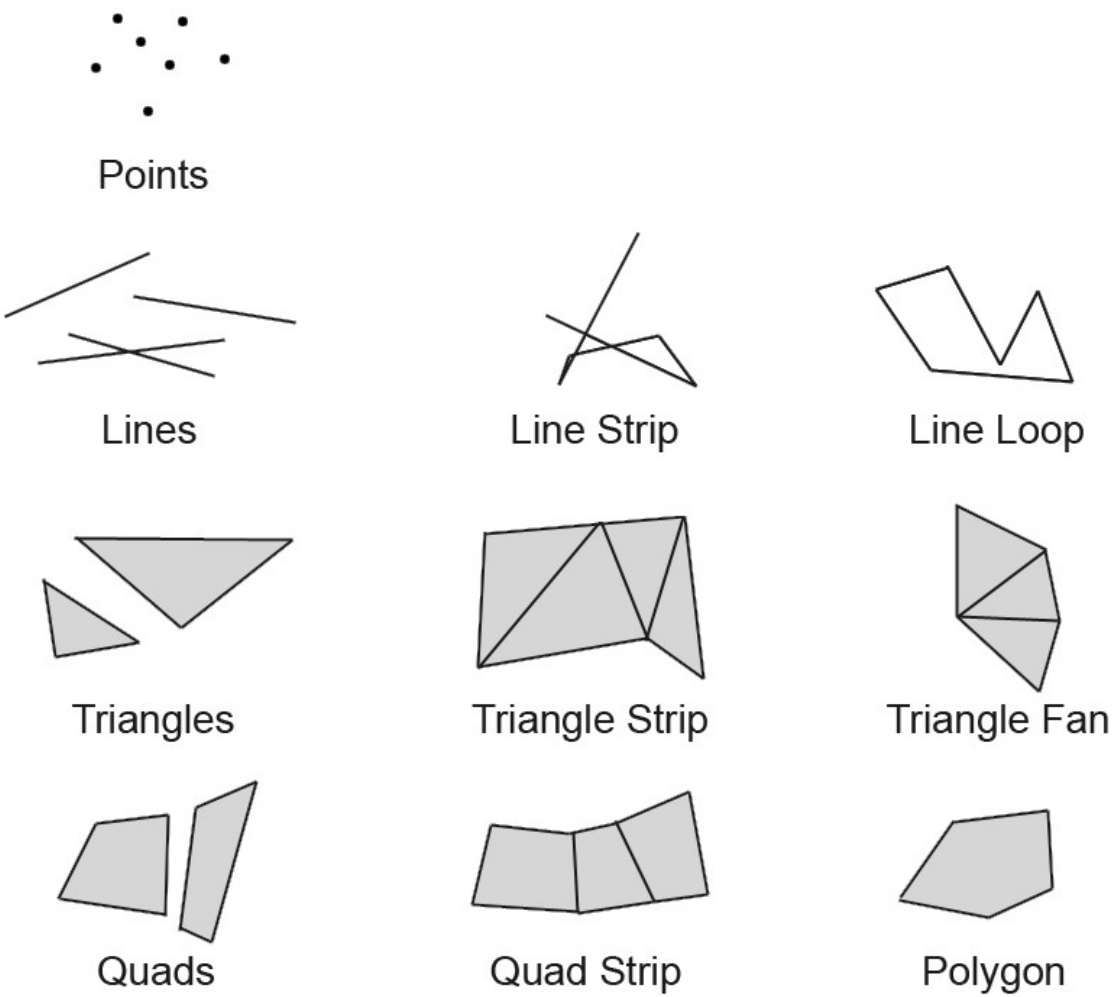
Graphic Primitives

For geometry that is stored on a GPU to be rendered by that GPU in real-time it must be stored such that it can be accessed and processed in an efficient time saving format.

To enable this a limited set of geometry types are supported:

- Typically geometry is comprised of points, lines, triangles and quadrilaterals (quads)
- These are the atomic units with which a 3d object is created, and are referred to as graphic primitives

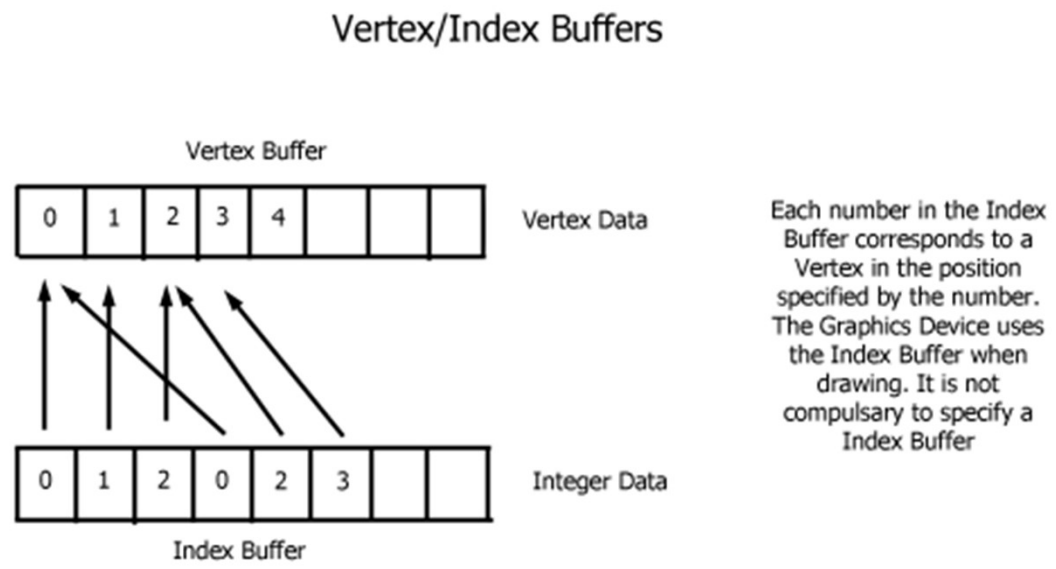
Graphic Primitives



Graphic Primitives

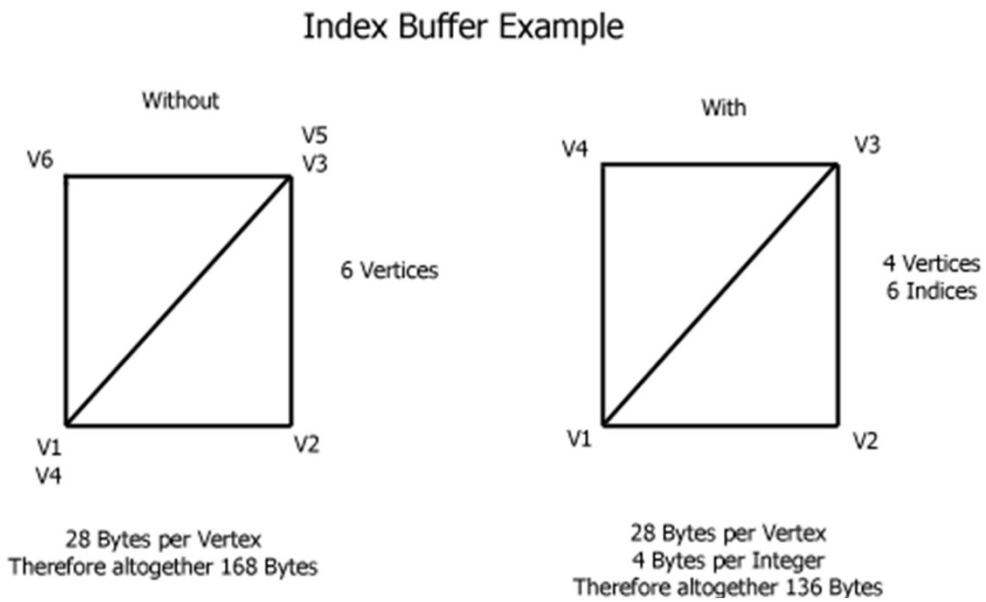
To provide for the description of complex object, graphic primitives are grouped together into a sequential series of the same type and stored in a vertex buffer in the GPU.

A more flexible and often efficient storage format is:



Graphic Primitives

A comparison of vertex buffer versus vertex/index buffer clearly indicates how much storage space can be saved on the GPU, and also how much less data would be required to be “pumped” through the pipeline.



A properties index buffer is also provided and which identifies what should be drawn with the same material and texture, and at therefore together.

Graphic Primitives

To enable very large regular numbers of graphic primitives to be drawn at “one go” they are grouped together into a sequential series of the same type.

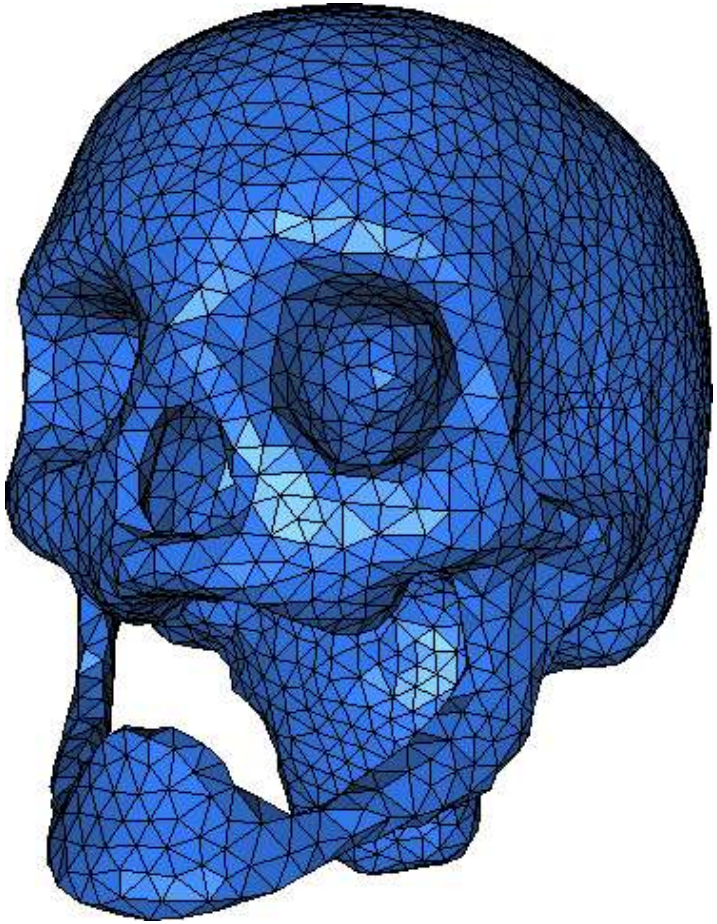
So:

- A non-regular polygonal surface can be represented by a vertex buffer grouped into triple vertex groups (this is a very **inefficient** way to store and draw an object).
- A non-regular polygonal surface can be represented by a vertex buffer referenced by an index buffer (this is a very **efficient** way to store and draw an object).

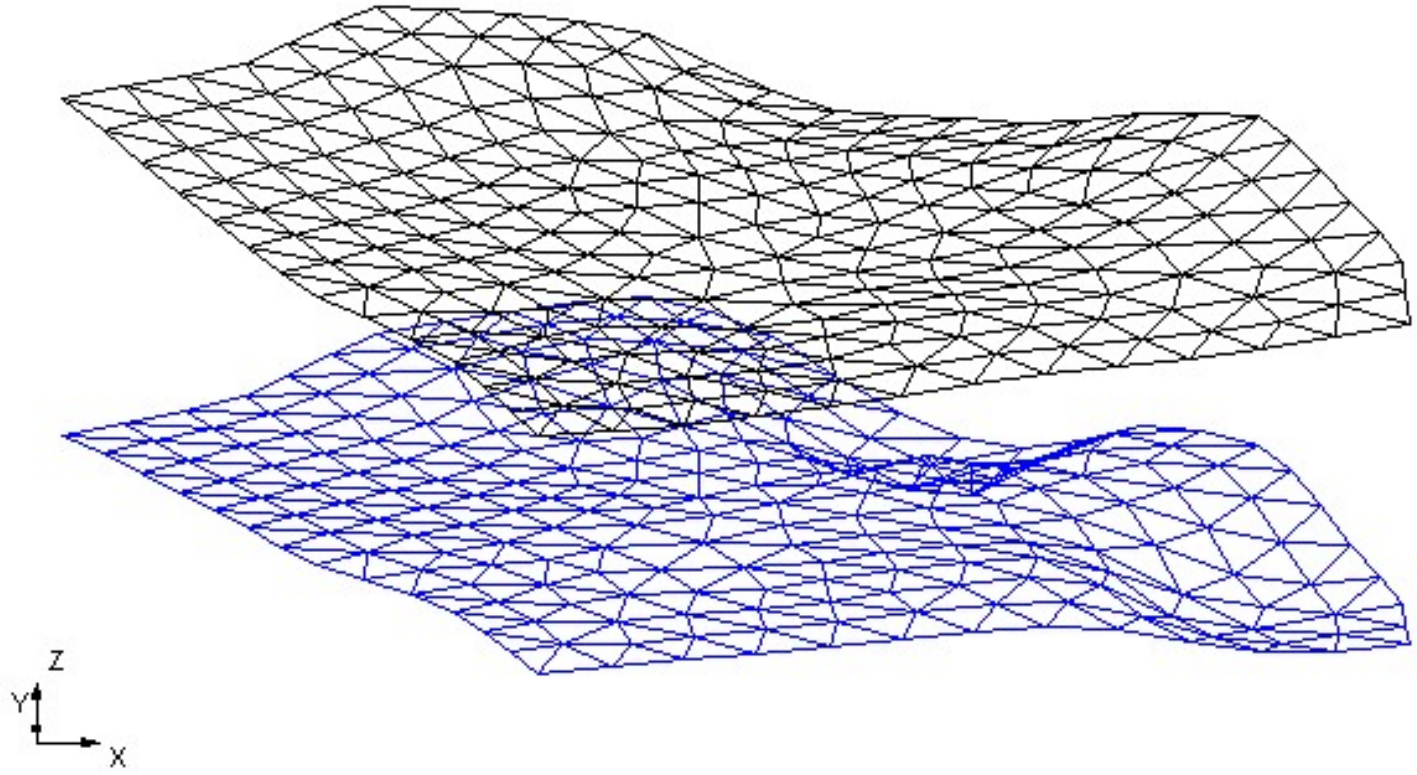
And:

- A parametric surface – such as a NURB – can be represented by a regular polygonal mesh stored as a series of line strips - but this is an approximation.

Graphic Primitives – non regular polygonal object



Graphic Primitives – regular polygon mesh



Rendering a 3D scene

Requires:

- Geometry – graphic primitives

And:

- Material properties of the geometry – ambient, diffuse, specular
- Lights to illuminate the geometry – point, spotlight, area
- Textures – one or more layers of images to “glue” onto the geometry
- Virtual camera

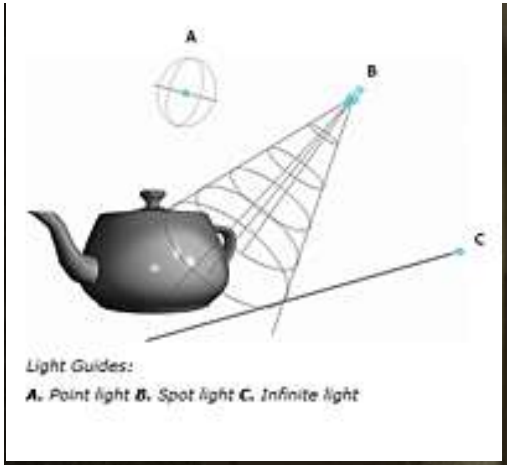
Materials

Materials can be applied to all graphic primitives.

Materials

Materials can be applied to all graphic primitives.

Lighting



Light Guides:
***A.** Point light **B.** Spot light **C.** Infinite light*

Textures

Textures can be applied to all graphic primitives.

Typically it is most obvious when applied to surfaces.

Surfaces are represented by triangles – a quad is deconstructed within the GPU into 2 triangles.

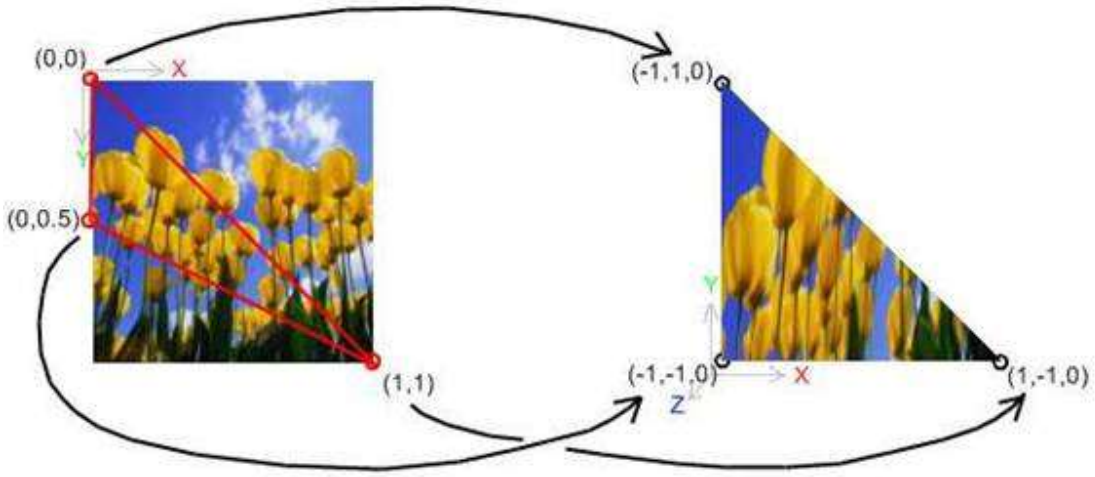
Textures

Each triangle consists of 3 vertices each which are represented by:

- Vertex – 3D position represented by 4(!) floating point values

And optionally:

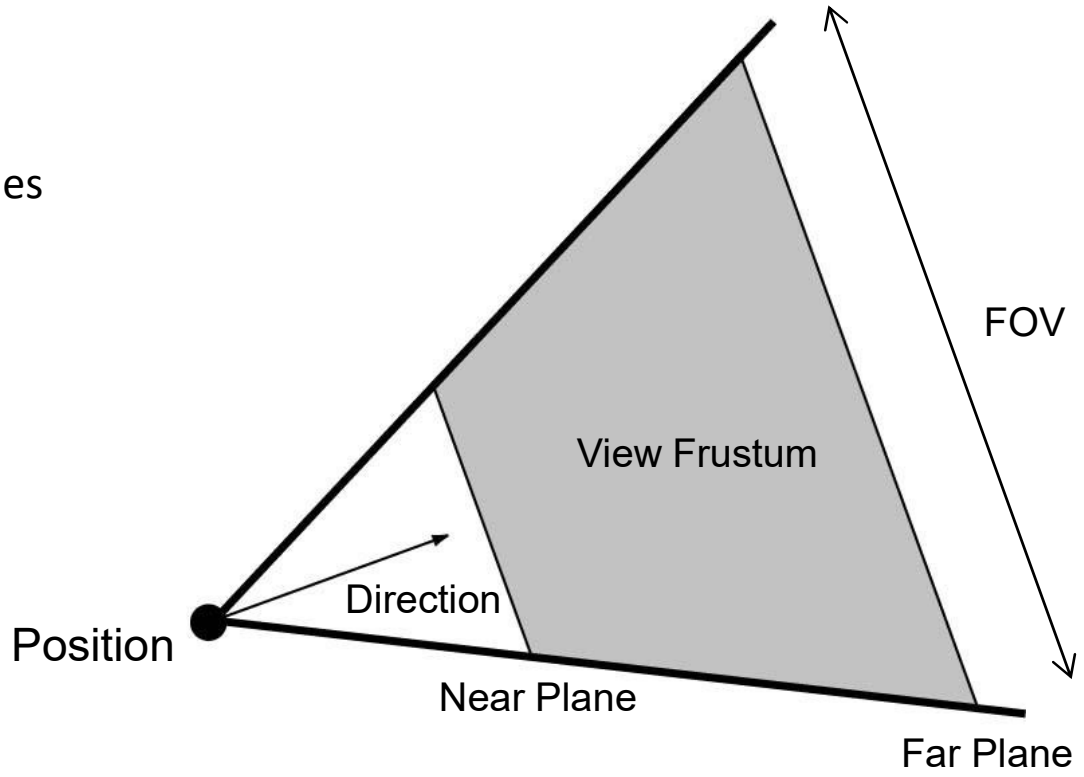
- Diffuse value - RGBA
- Surface unit-normal – XYZ
- Multiple texture UV 2D coordinates – diffuse, specular, bump, displacement



Virtual Camera

Each camera (sometimes referred to as a Point Of View (POV) is defined by:

- 3D position
- 3D direction vector
- 3D up vector
- Field of view (FOV)
- Near and far clipping planes



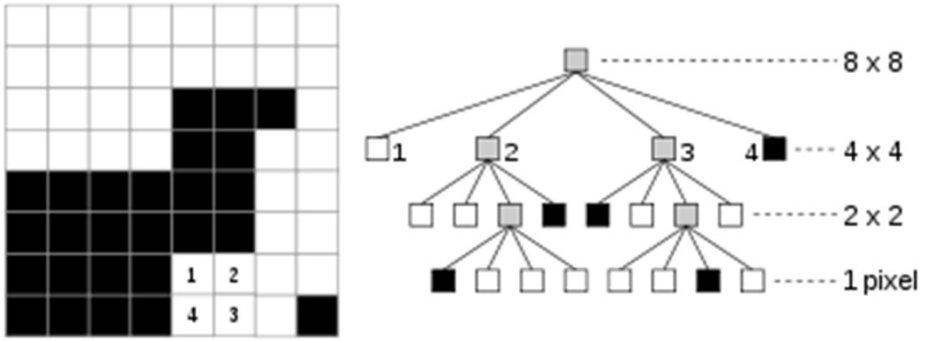
The Application Stage

This is executed entirely on the CPU and:

- requires that the programmer decides what happens when and where

Advanced examples of CPU based programming include:

- Animation
- Collision detection
- Speed up techniques – optimized spatial portioning algorithms such as Quad and Octree



The Application Stage

Important tasks include:

- Set up the Application at run-time:
  - Initialising
  - Creating or loading from disk, and then,
  - Sending graphic primitives to be stored on the GPU
  - Laying out the Graphical User Interface (GUI)

- Displaying the Application:
  - Updating the 3D scene
  - Ordering how the 3D models are drawn with the 3D Scene
  - GUI
  - Managing Input

The Application Stage

Important tasks also include:

- Clean up the Application when exited in an orderly fashion:
  - Providing for an exit dialogue (YES/NO)
  - Save “state of play”

Also, and discussed in later lectures:

- Cleanup or reset and initialise - when an Application has been put to sleep or has lost context – data that is stored on a GPU (i.e. geometry, lights, textures).
- Set up - when an Application has been put to sleep or has lost context – the data that is stored on a GPU.

The Geometry Stage

From within an application and via API calls order “geometrical” operations, which are executed on the GPU and effect the graphic primitives stored on the GPU and which are drawn after such operations.

Allows:

- Moving objects (matrix multiplication)
- Moving the camera (matrix multiplication)
- Setup lighting at vertices of triangle
- Project onto screen (3D to 2D)
- Clipping (avoid computation on triangles outside screen)
- Map to window

Animation within the Geometry Stage

An application can use “geometrical” operations to perform animation in many different ways.

Example:

- Before displaying a torus on screen, a matrix that represents a rotation can be applied.
- The result is that the torus is rotated.

This same process can also be applied to the animation of a virtual camera (this is possible since motion is relative).

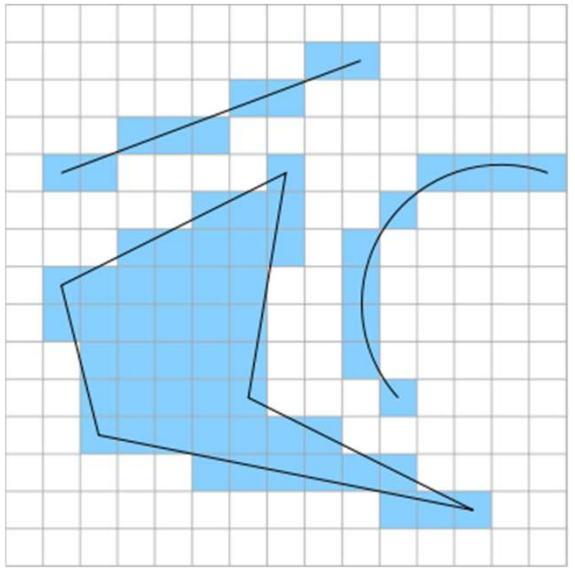
Rasterizer Stage

Rasterization – a 3D scene is rendered into an image using the GPU.

- In this the main task is to take graphic primitives, in a sequence determined by previous calls by the API, and turn these into visible coloured pixels in a window on a screen

- Add textures and various other per pixel operations

- Visibility is resolved using a Z Buffer



End of PART 1:

Mathematics for Computer Graphics