DM 2231
GAMES DEVELOPMENT TECHNIQUES

2015/16 SEMESTER 1

Week 1 – 3D Game Development

MODULE REQUIREMENTS

- You already have learnt C++ and OpenGL.
- You already have completed Studio Projects 1 and 2
  - Game Development Techniques will teach you the finer details about game development
    - How to manage data and display
    - How to create extensible programming framework
    - etc

MODULE SCHEDULE

Week	Dates	Topic	Remarks	Public Holidays
1	20-Apr-2015 to 24-Apr-2015	Module Introduction / 3D Game Programming	Issue Assignment 1	
2	27-Apr-2015 to 1-May-2015	Game Application		1 May. Labour Day
3	4-May-2015 to 8-May-2015	User Input		
4	11-May-2015 to 15-May-2015	Camera and GUI #1		
5	18-May-2015 to 22-May-2015	Camera and GUI #2		
6	25-May-2015 to 29-May-2015	Basic Game Physics		
7	1-Jun-2015 to 5-Jun-2015	Implementing Game Audio (E-learning)	Submit Assignment 1	1 Jun. Vesak Day
8	8-Jun-2015 to 12-Jun-2015	Mid-Sem Break		
9	15-Jun-2015 to 19-Jun-2015	Mid-Sem Break		
10	22-Jun-2015 to 26-Jun-2015	2D Game Programming #1	Issue Assignment 2	
11	29-Jun-2015 to 3-Jul-2015	2D Game Programming #2		
12	6-Jul-2015 to 10-Jul-2015	2D Game Programming #3		
13	13-Jul-2015 to 17-Jul-2015	Game Data		17 Jul. Hari Raya Puasa
14	20-Jul-2015 to 24-Jul-2015	Design Pattern #1		
15	27-Jul-2015 to 31-Jul-2015	Design Pattern #2		
16	3-Aug-2015 to 7-Aug-2015	Basic Artificial Intelligence (E-learning)		7 Aug. SG50 Public Holiday
17	10-Aug-2015 to 14-Aug-2015	Good Programming Practices	Submit Assignment 2	10 Aug. National Day

REFERENCE TEXT (AT NYP LIBRARY)

- Core techniques and algorithms in game programming, Dalmau, Daniel Sanchez-Crespo, New Riders Games (September 21, 2003)

TABLE OF CONTENT

- 3D Game Development
  - Introduction
  - Fundamental Data Types
  - Geometry Formats
  - A Generic Graphics Pipeline

3D GAME DEVELOPMENT

- Introduction
  - Hardware improvements allows graphics engines to evolve from simple sprite painting routines to real-time geometry processors
  - Previously, focus was on virtual reality at interactive frame rates
  - Currently, focus is on creating the illusion of immersion

3D GAME DEVELOPMENT

- The most important feature in Games Development (IMHO)

FRAME RATE

- 3D Real-time Games
  - Frame rate is vital for smooth gameplay
    - Many entities to compute and display will slow down frame rate
    - Use various techniques to display entities without slowdown
  - Bad frame rate == turns gamers off
  - You can develop **complex & advanced algorithms**, but if it causes **lag**, that is **bad**!

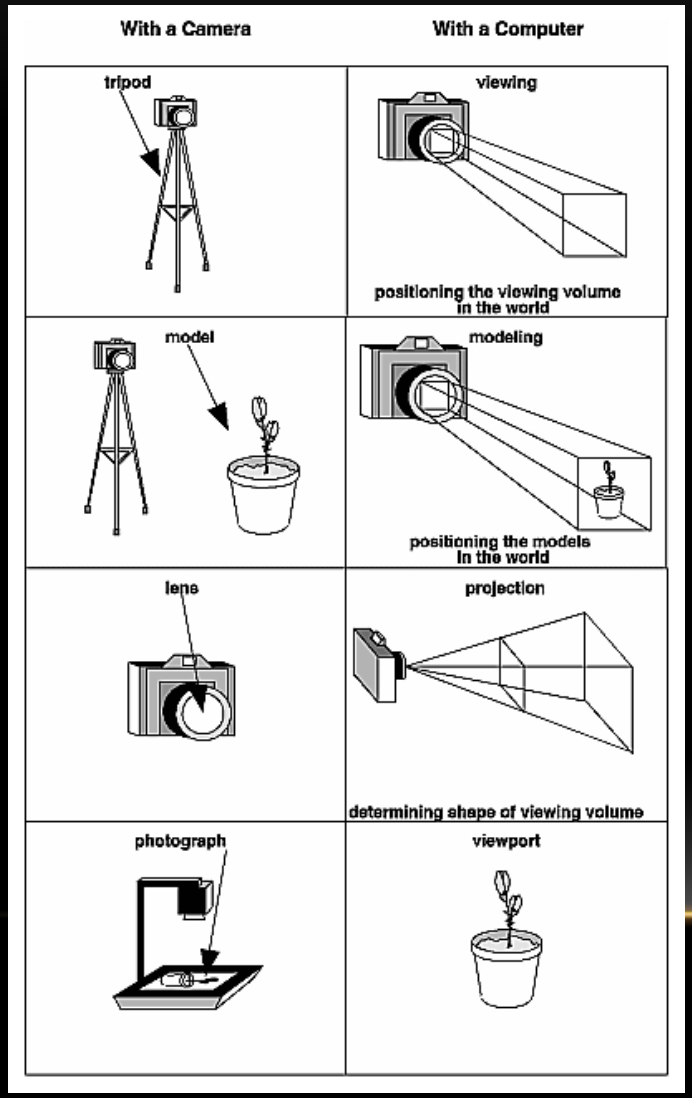
3D GAME DEVELOPMENT

- In this lecture, we will …
  - Look at the design and coding of 3D game pipelines
  - Analyze the global data structures and algorithms
    - to deliver the geometry stream to the 3D hardware efficiently.
  - View a global pipeline framework
    - which we can use to build a wide range of simple 3D games.

INTRODUCTION TO 3D AND DATA MANAGEMENT

- 3D Computer Graphics
- 3D Coordinate Systems
- Fundamental Data Types: Vectors and Coordinates in 3D
- 3D Object Representation
- Fundamental Data Types
- 3D Graphics Pipeline

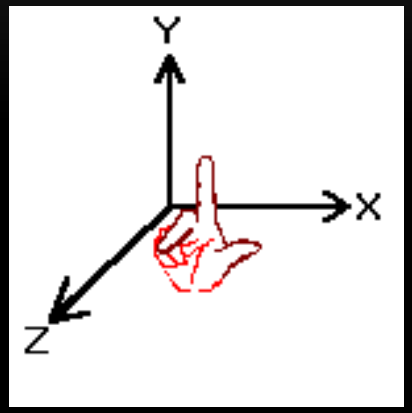
3D COMPUTER GRAPHICS

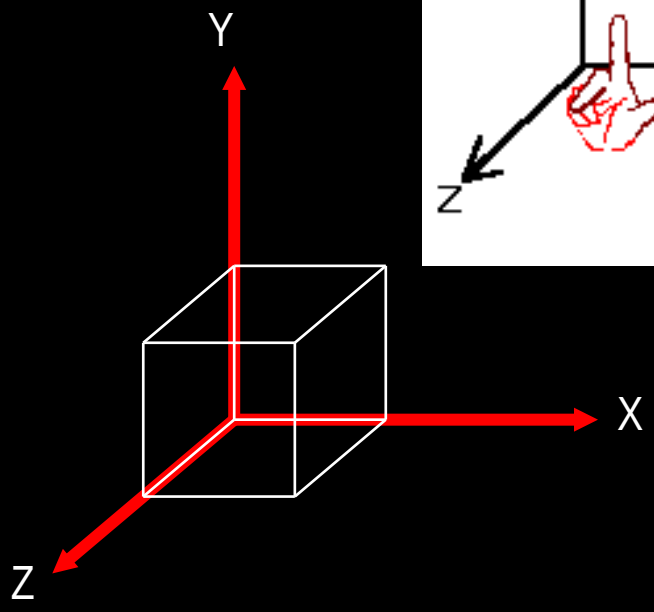


- Mathematical representation to create illusion of 3D
- Data structure to store 3D data is different from 2D
  - Mathematical operations is more complex for 3D compared to 2D

3D COORDINATE SYSTEMS

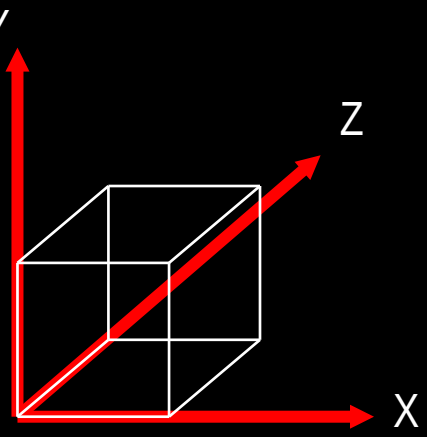
Right-Hand Coordinate System
OpenGL and XNA use this!





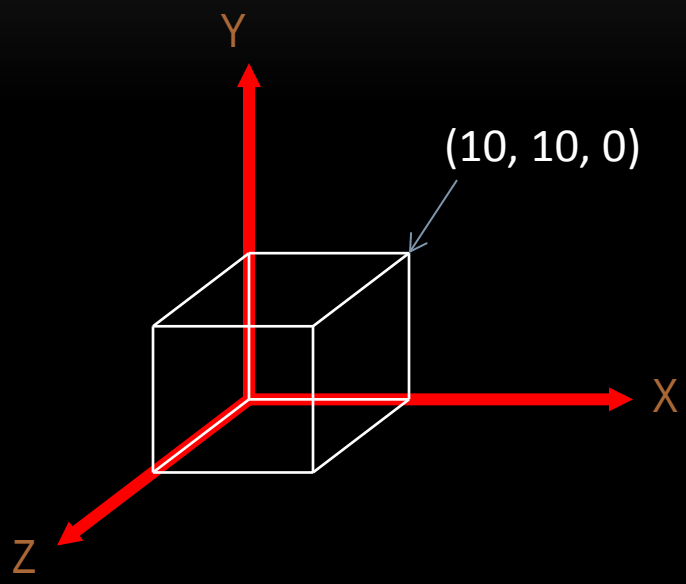
Right-Hand
Coordinate System

Left-Hand Coordinate System
Direct3D uses this



Left-Hand
Coordinate System

FUNDAMENTAL DATA TYPES: VECTORS AND COORDINATES IN 3D

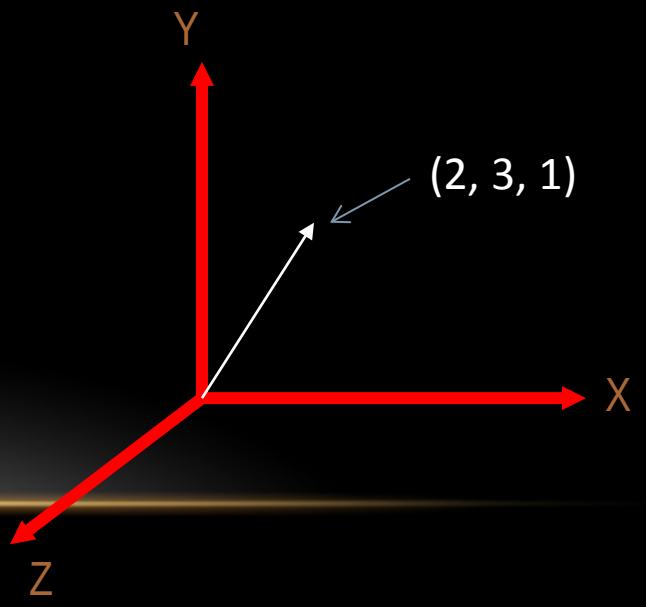


Vertex / Vertices

- 3 Numbers!
- Represent a location in space
- Often called a *point*, *position*, or *coordinate*

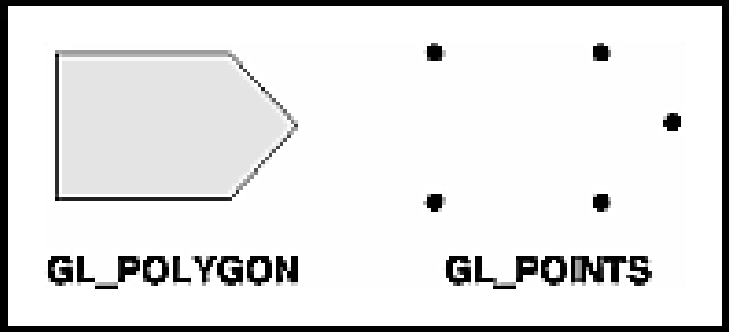
Vector

- Can represent a direction or a position

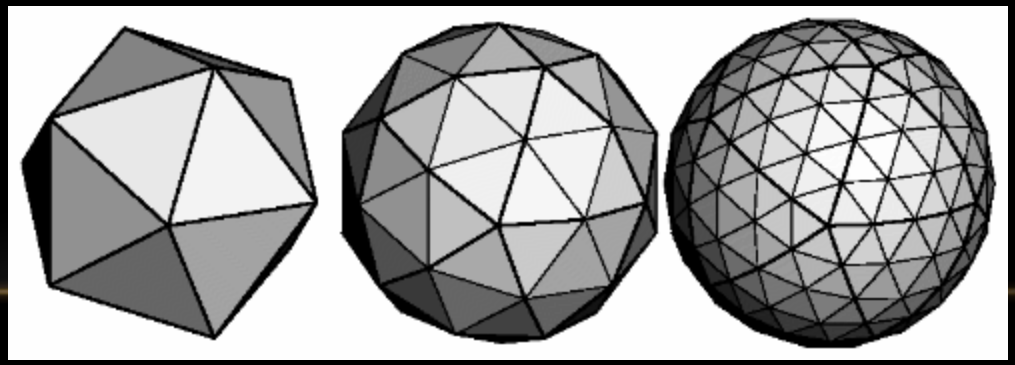


3D OBJECT REPRESENTATION

- A set of vertices can represent a triangle or a quad

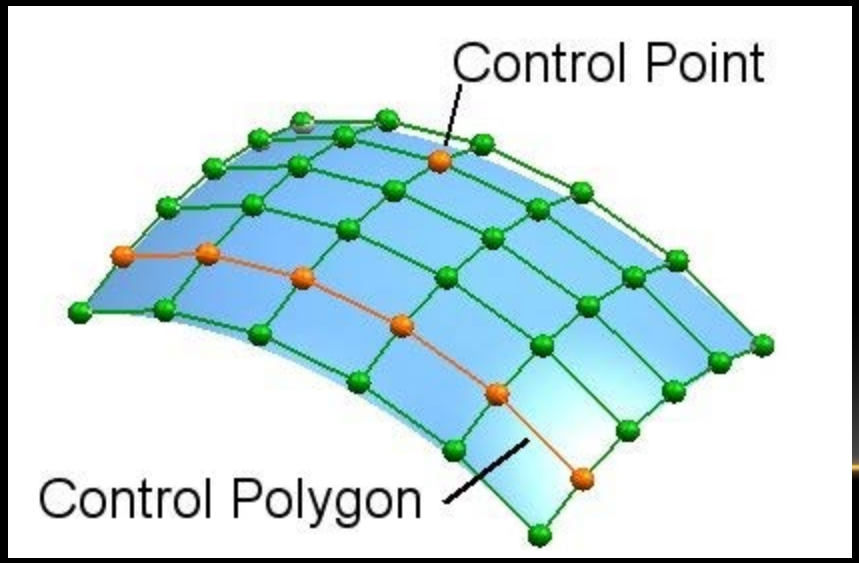


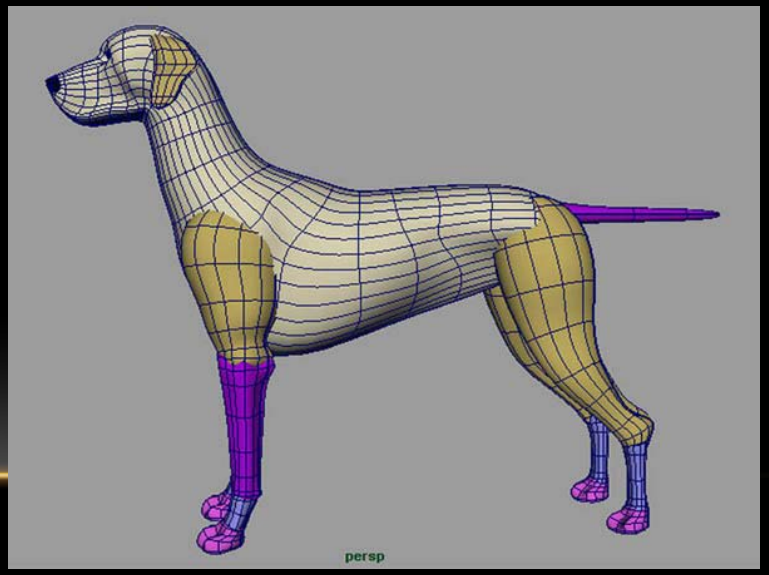
- A set of triangles or quads can represent a 3D Object.



3D OBJECT REPRESENTATION

- More triangles or quads in a 3D Object
  - Gives you a more accurate representation of the **actual/ideal** object
- How about NURB surfaces?

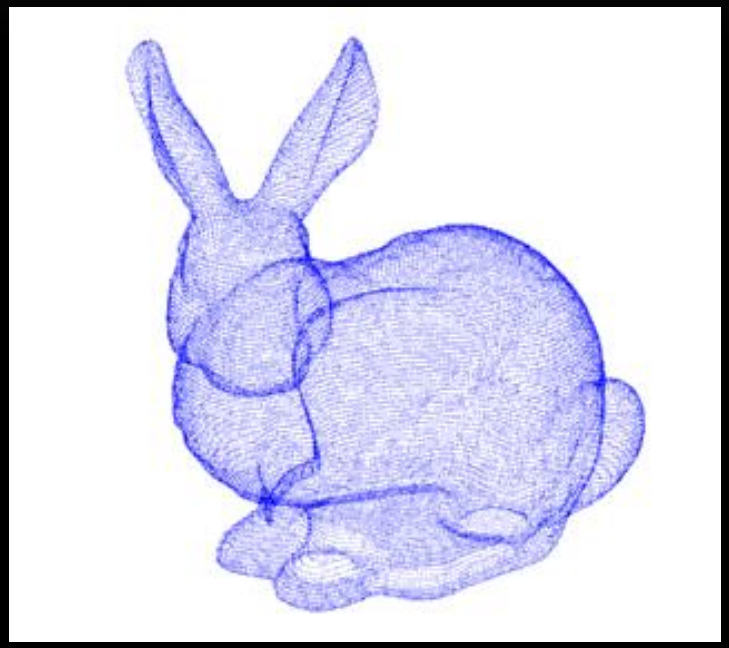


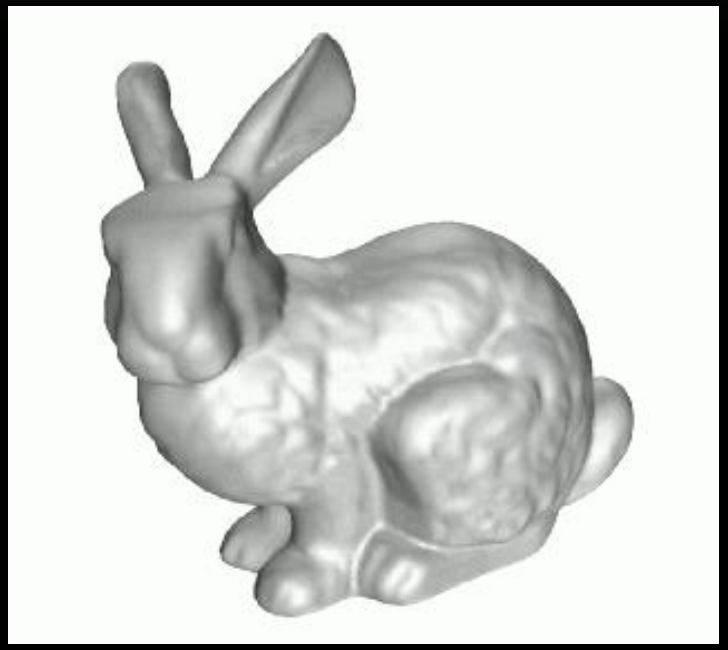


3D OBJECT REPRESENTATION

- NURB are smooth interpolation surfaces
  - OpenGL triangulates the NURB surfaces into little triangles within a tolerance.
  - Drawback of more triangles == more computation needed == slower display == lower frames per second (fps)
    - This will affect your game's gameplay, playability and fun.
- How about using Point Clouds?

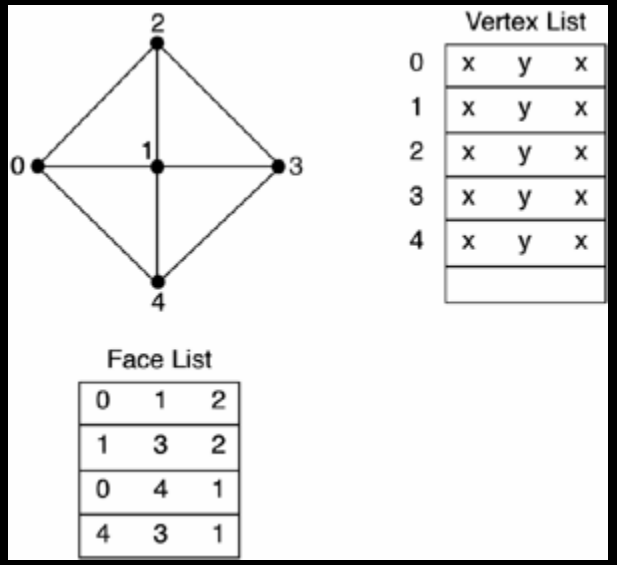
3D OBJECT REPRESENTATION





FUNDAMENTAL DATA TYPES: INDEXED PRIMITIVES

- What if you have vertices which are shared by several triangles?
  - Use indexed primitives!
- indexed primitives has 2 lists: Vertex and face lists
  - Vertex List: Stores unique vertices
  - Face List: The index of vertices to form each triangle



FUNDAMENTAL DATA TYPES: INDEXED PRIMITIVES

- Advantage:
  - Lower memory usage
  - Faster data transfer from system memory to graphics card through the system bus
  - Modern Graphics cards support index primitives
    - Improved performance at zero CPU cost!

- Disadvantage :
  - How to do texture mapping?

FUNDAMENTAL DATA TYPES: QUANTIZATION

- Reduce memory usage at expense of accuracy
- Keep the data compressed in memory, and uncompress it whenever you need to use it.
  - this specific compression technique is so fast that it can be used in real-time applications.
  - Store values in lower precision data types, thus saving precious space in the process.

Example: Downsample a double to a float

Reduce memory use by 50%!

Lose precision!

Use only when the end result is negligible!

- Common use is with coding floats into unsigned shorts (16 bits) or even unsigned bytes (8 bits).

FUNDAMENTAL DATA TYPES: QUANTIZATION

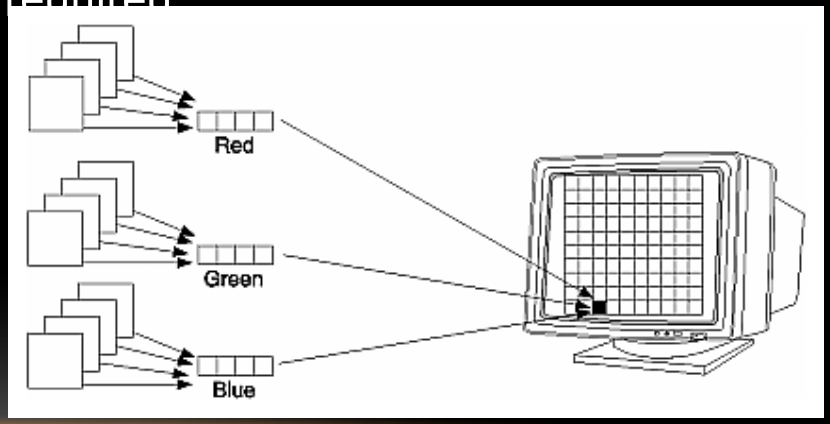
- Example: Assume you are storing the position of a player within an environment of [0..1000] X [0..1000] units. Store the position of (500,500) in **float** vs quantised storage using **short**.
  - Float: total of 64-bits to store two 32-bit values.
    - 1 bit for the sign(s)
    - 8 bits for the exponent (e)
    - 23 bits for the mantissa (m)
  - Quantised storage: total of 32-bits to store two 16-bit values.

$$CompressedValue = CompressedTypeSize \cdot \left(\frac{OriginalValue - MinValue}{MaxValue - MinValue} \right)$$

$$= 2^{16} \cdot \left(\frac{OriginalValue - 0}{1000 - 0} \right) = 65536 \cdot \left(\frac{500 - 0}{1000 - 0} \right) = 32768$$

FUNDAMENTAL DATA TYPES: COLOUR

- 3D games mainly use the RGB (or RGBA) color space
  - Usually use float
    - But are the range and precision of float needed?
  - Bytes are more suitable?
    - internally supported by most APIs and GPUs, so no decompression/reconstruction is required
- BGR (or BGRA) format?
  - Targa texture format?



FUNDAMENTAL DATA TYPES: ALPHA

- Encodes transparency in vertex or pixels
- Use with colour values to form 32-bit RGBA values
- The lower the value, the less opacity:
  - Zero alpha would be completely invisible.
  - Alphas closer to 1 (in floats) or 255 (in bytes) designate opaque
- Using alpha values in texture maps
  - Make your textures grow by 25%!!!
- If you need to do a texture mapping with constant alpha (such as a glass window, for example), you can save precious memory by using a regular RGB map and specifying alpha per vertex instead.
- Use alpha values in a texture map whenever it's absolutely necessary.

3D GRAPHICS PIPELINE

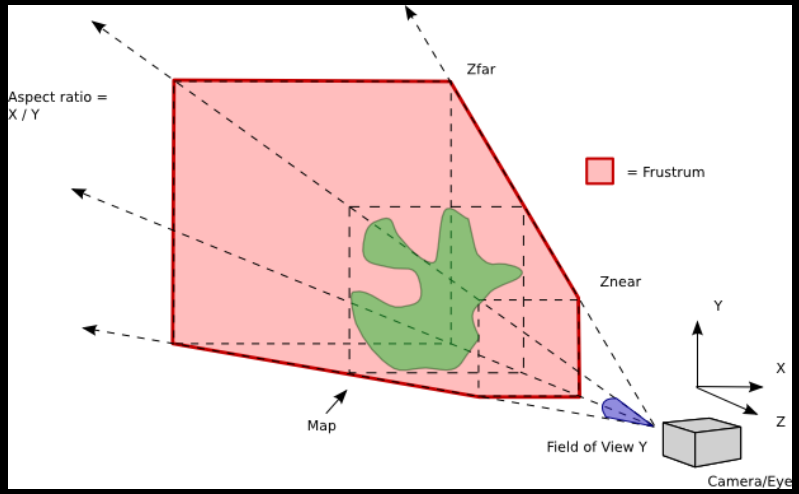
- Four stages
  - Visibility determination
    - Clipping
    - Culling
    - Occlusion testing
  - Resolution determination
    - LOD analysis
  - Transform, lighting
  - Rasterization

3D GRAPHICS PIPELINE: CLIPPING

- Removing unseen / unwanted entities from the display
  - Check it against a clipping volume, such as the screen.
  - If check fails, entity is not displayed as we are sure it is not seen.
- Advantage:
  - Skip the processing of the data. Will it improve framerate?
- Drawback:
  - The clipping test must be faster than drawing the entities in order for clipping to provide a significant speedup.
  - Else clipping would be an unneeded burden.

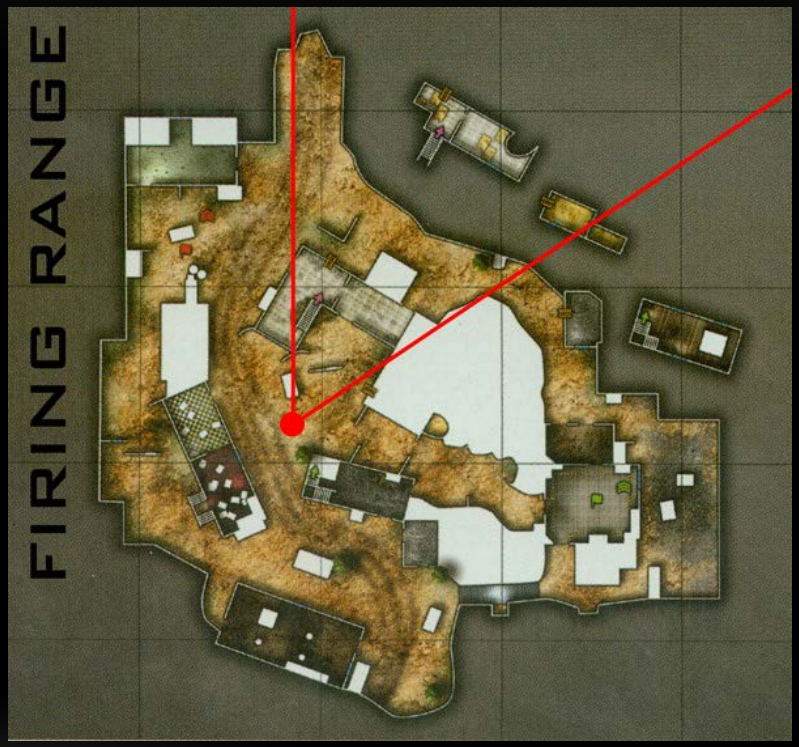
3D GRAPHICS PIPELINE: CLIPPING

- Removing unseen / unwanted entities from the display
  - Check it against a clipping volume, such as the screen.
  - If check fails, entity is not displayed as we are sure it is not seen.
- Advantage:
  - Skip the processing of the data. Will it improve framerate?



3D GRAPHICS PIPELINE: CLIPPING

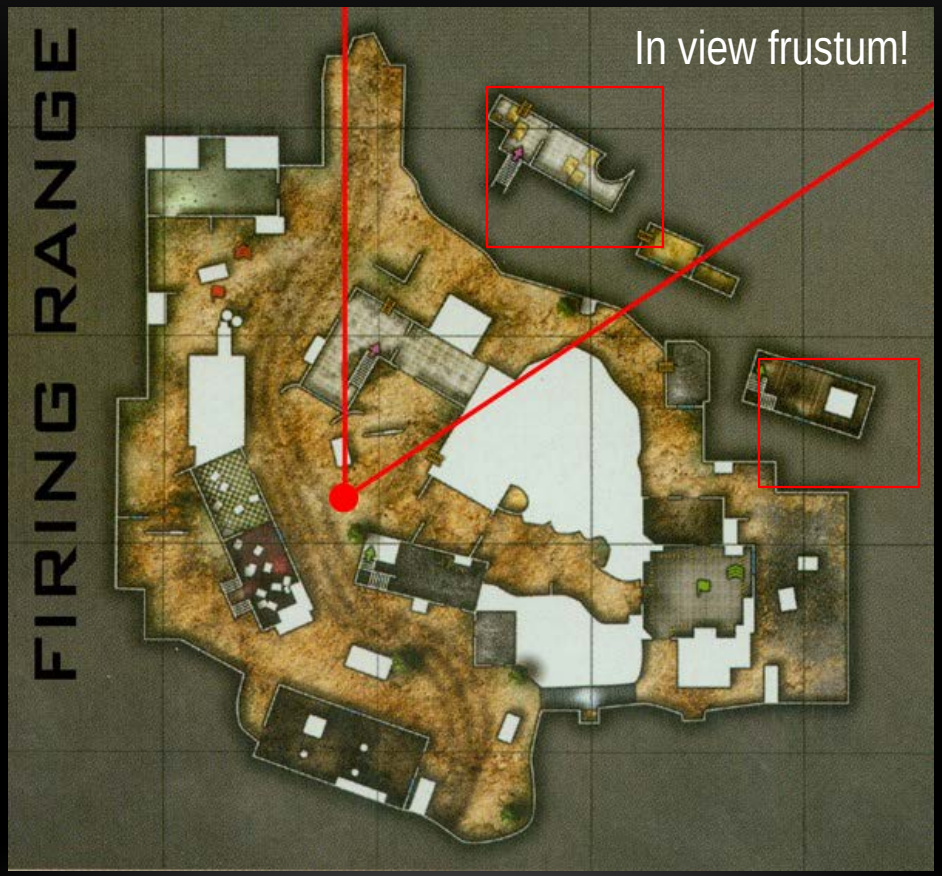
- Drawback:
  - The clipping test must be faster than drawing the entities in order for clipping to provide a significant speedup.
  - Else clipping would be an unneeded burden.
- Generally, it is good to clip geometry early in the graphics pipeline
  - Reduce data transfer to the GPU, and ultimately, the rasterization of unseen entities
  - Use Scene Management?



3D GRAPHICS PIPELINE: OBJECT CLIPPING

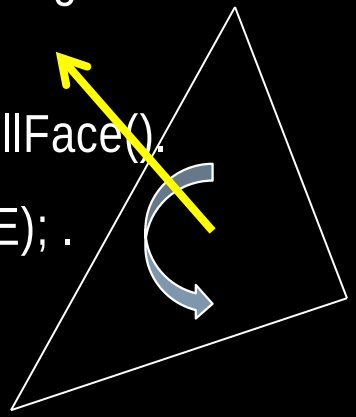
- Avoid displaying an object completely
  - The object's data is not sent to the Graphics Card at all
  - Scene Management is used
    - Decompose our scene into a list of objects
    - Test each object for clipping, as it was an entity
      - Use bounding box? Is it always accurate?
    - if a whole object is out of clipping volume, then avoid sending the data to graphics card.
    - Else send the data and let hardware triangle clipping do its work.
  - Advantage: Can avoid clipping tests for thousands to millions of entities!

3D GRAPHICS PIPELINE: OBJECT CLIPPING

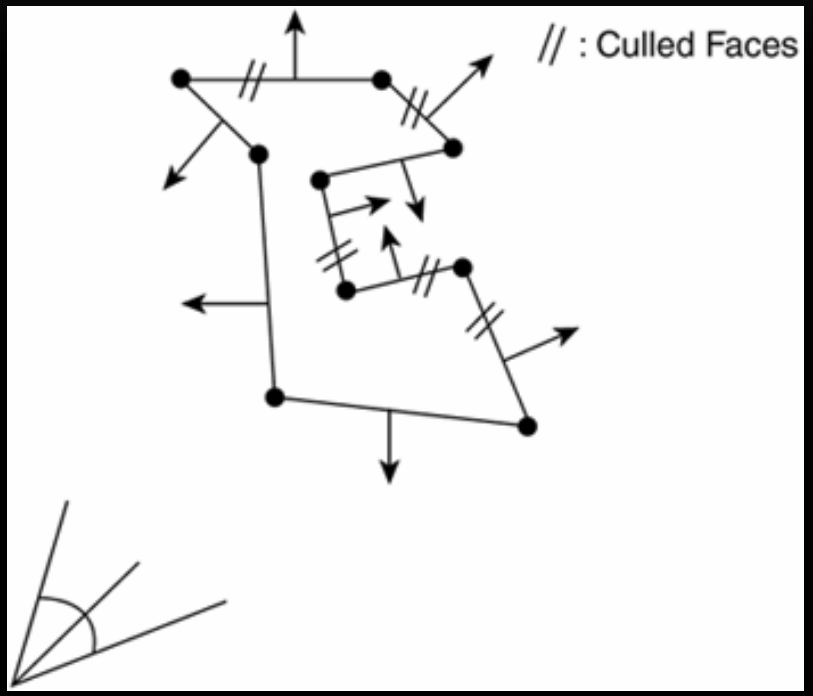


3D GRAPHICS PIPELINE: FACE CULLING

- Avoid displaying the triangles which are not facing the camera
  - Reduces the processing of triangles by 50%?
  - OpenGL face culling displays a triangle when the window coordinates of a triangle are in a counter-clockwise order.
    - Use glFrontFace() to specify the ordering, counter-clockwise or clockwise, to be interpreted as a front-facing or back-facing primitive.
    - Specify culling either front or back faces by calling glCullFace().
    - Enable face culling by calling glEnable(GL_CULL_FACE); .



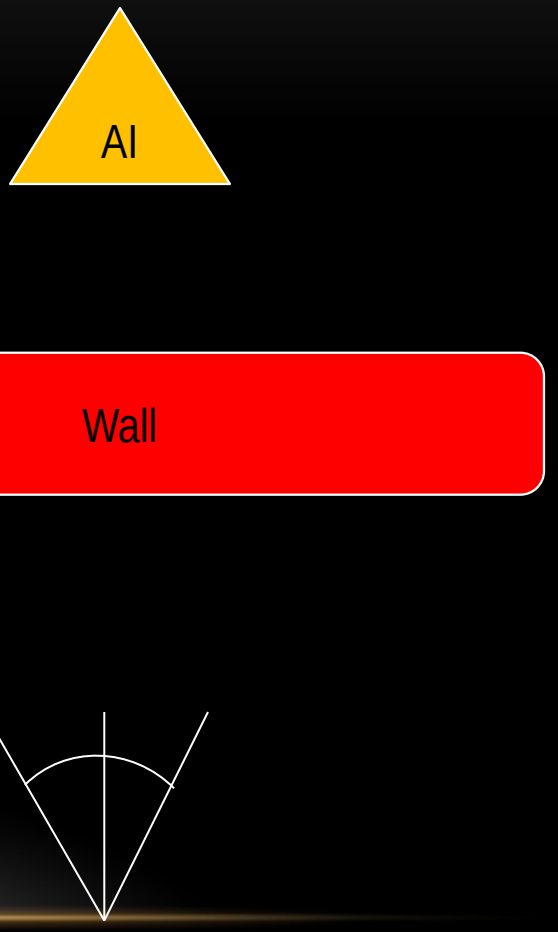
3D GRAPHICS PIPELINE: FACE CULLING



- Object-level culling less popular than object-level clipping
  - Benefit for culling vs clipping
    - 50% vs 80%
  - Culling is significantly harder to do.
  - If your geometries are prepacked in linear structures (vertex arrays or buffers), you don't want to reorder it because of the culling.

3D GRAPHICS PIPELINE: OCCLUSION

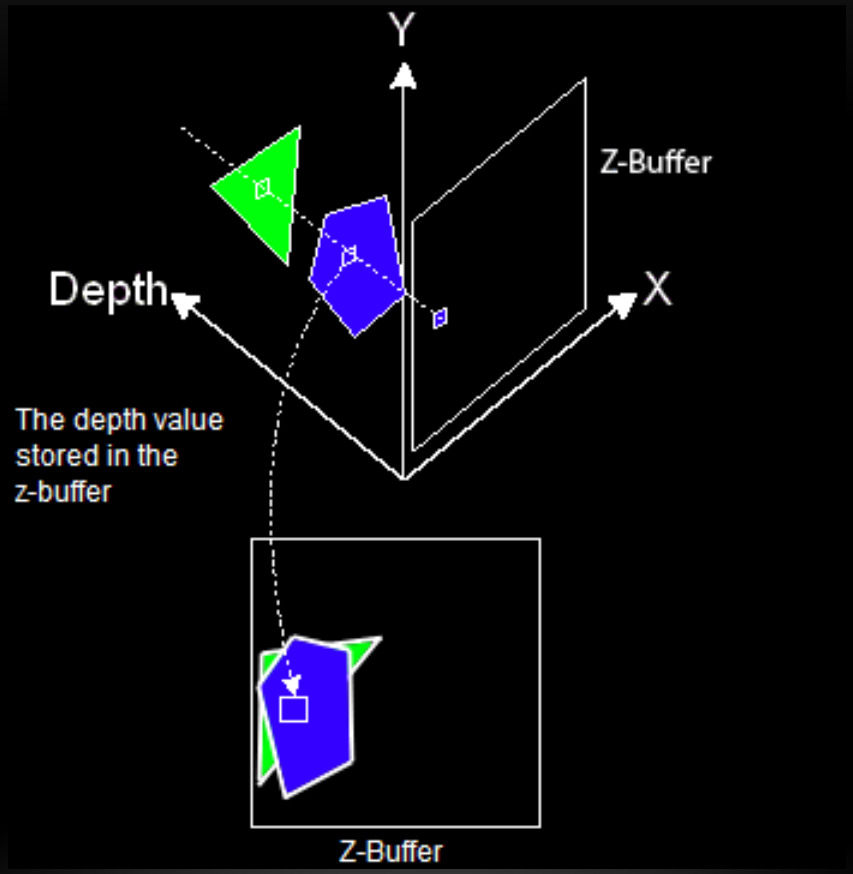
- What if you have 2 objects in your viewing frustum, and one is completely hidden behind the other?
- Use Occlusion Testing
  - Occlusion testing uses the graphics card to do testing



3D GRAPHICS PIPELINE: OCCLUSION

- The bounding volume of an object is sent to the graphics card.
  - Which tests the volume against the z-buffer
  - Retrieve from the graphics card
    - If the object actually modified the Z-buffer
    - If it did so, how many pixels were modified
  - Usually draw the front objects first, then rear objects.
  - Speedup graphics display
  - Good to use with scene management

3D GRAPHICS PIPELINE: OCCLUSION



3D GRAPHICS PIPELINE: RESOLUTION DETERMINATION

- What if you already have all these wonderful algorithms in place? But you are rendering a terrain which has millions or billions of polygons?
  - Even good algorithms cannot process so many polygons!
  - Either upgrade the hardware, especially the graphics card,
  - or reduce the number of polygons to process.
    - Will this affect the accuracy of the display?

3D GRAPHICS PIPELINE: RESOLUTION DETERMINATION

- Not if we reduce the number of polygons for selected entities!
  - Note that human visual system tends to focus on larger, closer objects (especially if they are moving)
- Two components
  - resolution-selection heuristic that allows us to assign relative importance to onscreen objects;
  - a rendering algorithm that handles the desired resolution.
  - More discussion on this topic in Advanced Game Development!

SUMMARY

- We have discussed about the main issues with 3D Game Development
  - Graphics for games are getting more complex
  - Smooth gameplay is dependent on frame rate
  - Various Game Development techniques to reduce
    - the memory usage
    - the computation and processing of the entities