# CSI 3202 Micro-Computer Graphics Z-Buffer Algorithm

Presenter: Girendra Persaud
University of Guyana

# Outline

- What is the Z-Buffering?
- The Visibility Problem
- Solving the visibility problem
  - The Painter's algorithm
  - Z-culling (Z-buffer algorithm)
- Steps to enable the algorithm in your 3D scenes
- Questions?
- Review Questions

# What is z-buffering?

- Z-Buffering is the management of depth coordinates in 3D graphics,
  - done for each object in the 3D scene
  - usually done in hardware
  - sometimes in software
- It is one solution to the visibility problem

* http://en.wikipedia.org/wiki/Z-buffering

# What is this visibility problem?

- The issue of deciding visibility between two objects

- How is this problem solved (geometrically)?
  - Given a set of obstacles in the space,
    - two points in the space are said to be **visible to each other**, if the line segment that joins them does not intersect any obstacles

- When projecting a 3D scene onto a 2D plane, it is necessary decide which polygons are visible, and which are hidden

# How the visibility problem in solved in 3D graphics

- There are several solutions to this problem, commonly you will find

- The painter’s Algorithm (Object-space Algorithms)

- Z-culling (Image-space Algorithms)

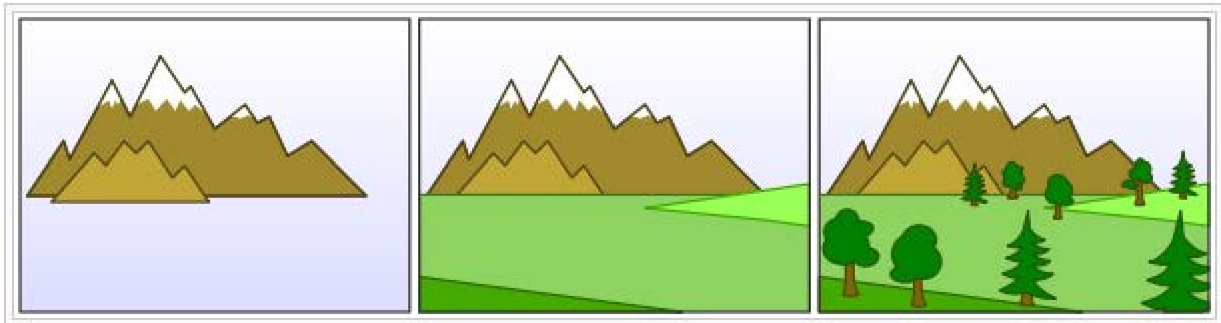
# The Painter’s Algorithm

- The **painter's algorithm**, also known as a **priority fill**, is one of the simplest solutions to the visibility problem in 3D CG
- the technique of painting distant parts of a scene before parts nearer (covering distant parts)

Foley, James; van Dam, Andries; Feiner, Steven K.; Hughes, John F. (1990). Computer Graphics: Principles and Practice. Reading, MA, USA: Addison-Wesley. p. 1174. ISBN 0-201-12110-7.

# The Painter's Algorithm

- The algorithm sorts all the polygons in a scene by their depth and then paints them in this order, farthest to closest

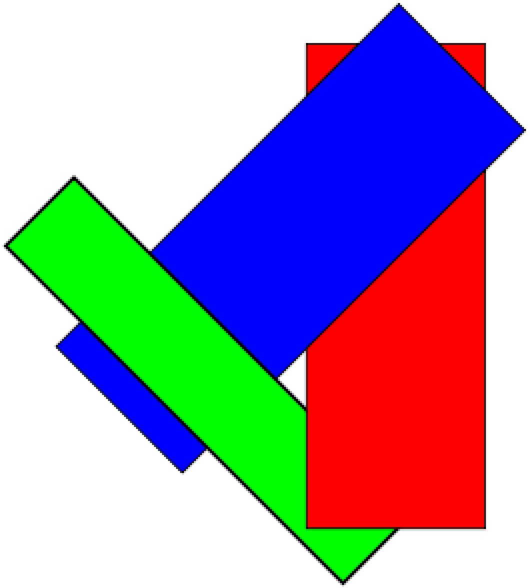


- See a problem?
  - Redundant painting

Foley, James; van Dam, Andries; Feiner, Steven K.; Hughes, John F. (1990). Computer Graphics: Principles and Practice. Reading, MA, USA: Addison-Wesley. p. 1174. ISBN 0-201-12110-7.

# The Painter's Algorithm

- Another Problem
  - it fails when there is overlapping polygons



Foley, James; van Dam, Andries; Feiner, Steven K.; Hughes, John F. (1990). Computer Graphics: Principles and Practice. Reading, MA, USA: Addison-Wesley. p. 1174. ISBN 0-201-12110-7.

# Z-culling

- Another solution to the visibility problem
- In rendering, z-culling is early pixel elimination based on depth, a method that provides an increase in performance when rendering of hidden surfaces is costly
- It is a direct consequence of z-buffering, where the depth of each pixel candidate is compared to the depth of existing geometry behind which it might be hidden
- When using a z-buffer, a pixel can be culled (discarded) as soon as its depth is known, which makes it possible to skip the entire process of lighting and texturing a pixel that would not be visible anyway

# Z-culling

- When an object is rendered
  - the depth of a generated pixel (z coordinate) is stored in a buffer (the **z-buffer** or **depth buffer**)
  - The buffer is arranged as a two-dimensional array (x,y) with one element for each screen pixel
  - If another object of the scene must be rendered in the same pixel, the algorithm compares the two depths and chooses the one closer to the observer
  - The chosen depth is then saved to the z-buffer, replacing the old one

# Z-culling - The Algorithm

- **Given:** A list of polygons {P1,P2,.....Pn}
  **Output:** A COLOR array, which display the intensity of the visible polygon surfaces.

- **Initialize:**
  note : z-depth and z-buffer(x,y) is positive........
  z-buffer(x,y)=max depth; and COLOR(x,y)=background color.

- **Begin:**

```
for(each polygon P in the polygon list) do {
    for(each pixel(x,y) that intersects P) do {
        Calculate z-depth of P at (x,y)
        If (z-depth < z-buffer[x,y]) then {
            z-buffer[x,y]=z-depth;
            COLOR(x,y)=Intensity of P at(x,y);
            } } }
```

# Z-Culling Maths

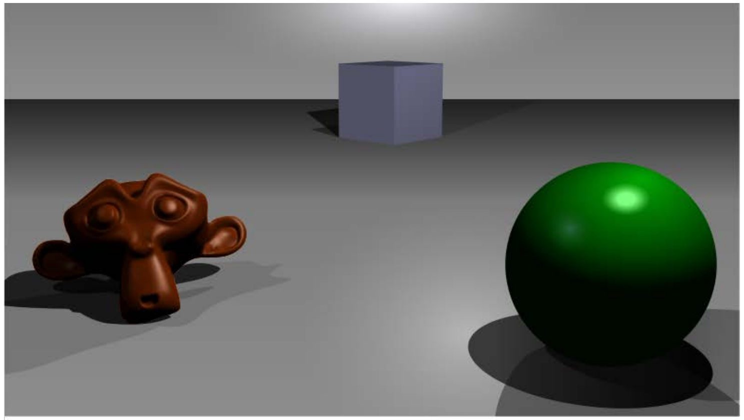
- The range of depth values in camera space is often defined between a *near* and *far* value of *z*
- After a perspective transformation, the new value of *z*, or *z*', is defined by:

$$z' = \frac{far + near}{far - near} + \frac{1}{z}\left(\frac{-2 \cdot far \cdot near}{far - near}\right)$$

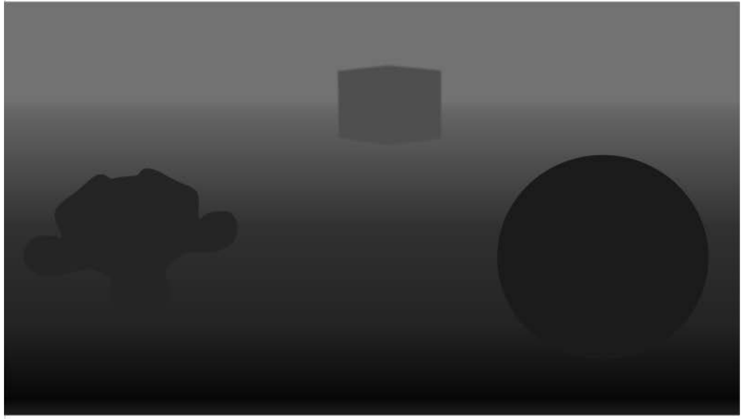
  - where *z* is the old value of *z* in camera space, and is sometimes called *w* or *w*'
  - The resulting values of *z*' are normalized between the values of –1 and 1, where the *near* plane is at

–1 and the *far* plane is at 1

# Z-buffer Representation



A simple three-dimensional scene



Z-buffer representation

# Z-buffer precision

- The granularity of a z-buffer has a great influence on the scene quality:
  - a 16-bit z-buffer can result in artifacts (called "z-fighting" or **stitching**) when two objects are very close to each other
  - A 24-bit or 32-bit z-buffer behaves much better, although the problem cannot be entirely eliminated without additional algorithms
  - An 8-bit z-buffer is almost never used since it has too little precision

# Steps to enable Depth Buffering in OpenGL

1. Ask for a depth buffer when you create your window.
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

2. Call glEnable (GL_DEPTH_TEST) in your program's init routine

3. Ensure that your zNear and zFar clipping planes are set correctly and in a way that provides adequate depth buffer precision

4. Pass GL_DEPTH_BUFFER_BIT as a parameter to glClear,
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

# Review

- What is the Z-Buffering?
- The Visibility Problem
- Solving the visibility problem
  - The Painter's algorithm
  - Z-culling (Z-buffer algorithm)
- Steps to enable the algorithm in your 3D scenes

# Questions?

# Review Question

- Pixel P = (1,2,26)
  - Near = 1, Far = 100
- Is pixel P inside the frustum?
- Show by way of calculations.