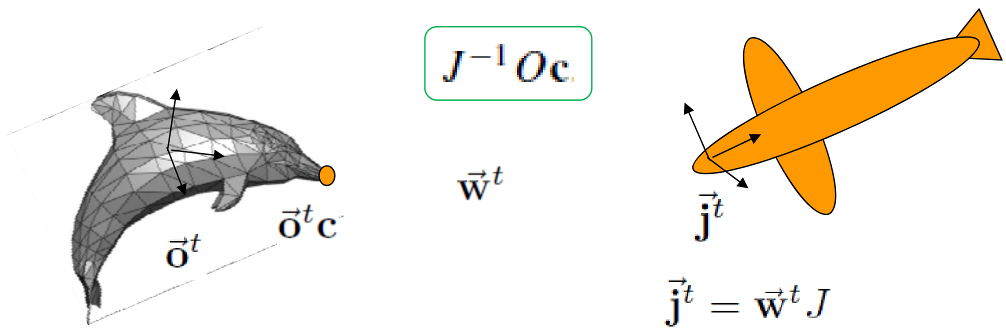
Hello World 3D

Chapter 6

Chapter 1~5 & 7~8 & 10~11

Quiz

- 5.1 Suppose we have a scene with a jet airplane flying in the sky. Suppose the geometry of the airplane is described with respect to the jet's own frame $\vec{\mathbf{j}}^t$, defined as $\vec{\mathbf{j}}^t = \vec{\mathbf{w}}^t J$. Let this frame be centered in the cockpit with its negative z axis facing out the front window. Suppose we wish to render the scene from the point of view of the pilot. Given a point on some other object: $\vec{\mathbf{o}}^t\mathbf{c}$, what is the coordinate vector that we should pass to the renderer to draw this point?



Modelview Matrix

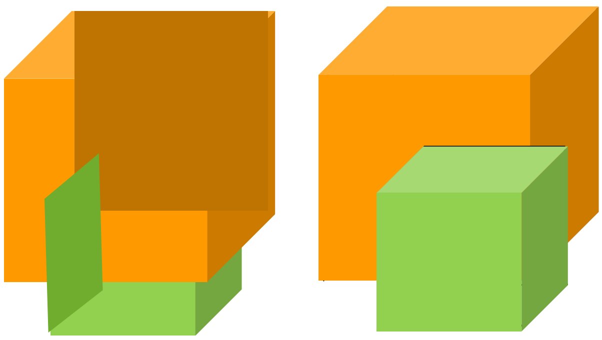
- Modelview matrix (*MVM*)
  - Describes the orientation and position of the view E^{-1} and the orientation and position of the object O with respect to the eye frame $\vec{\mathbf{e}}^t$

$$\tilde{p} = \vec{\mathbf{o}}^t \mathbf{c} = \vec{\mathbf{w}}^t O \mathbf{c} = \vec{\mathbf{e}}^t E^{-1} O \mathbf{c}$$

  - The vertex shader will take these vertex data and perform the multiplication $E^{-1}O\mathbf{c}$, producing <u>the eye coordinates</u> used in rendering

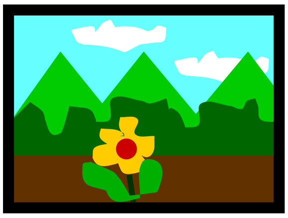
Drawing a shape

```
static void InitGLState(){
  glClearColor(128./255., 200./255., 255./255., 0.);
  glClearDepth(0.0);
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_GREATER);
  glEnable(GL_CULL_FACE);
  glCullFace(GL_BACK);
}
```

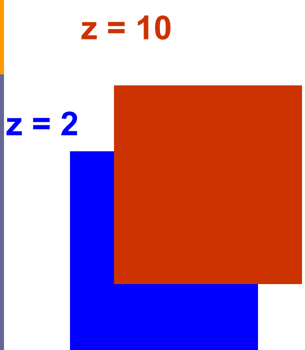


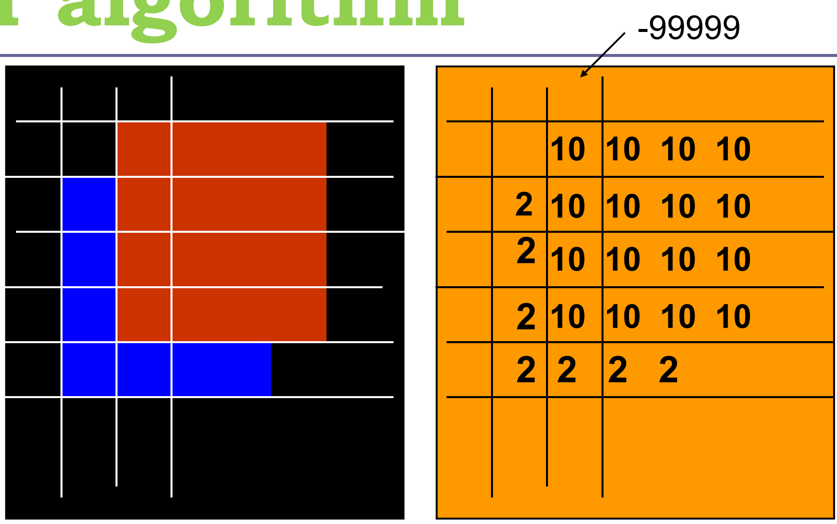
Hidden-Surface Removal

- 2 philosophies
  - remove the hidden surfaces (hidden surface removal algorithm)
  - select the visible surfaces (visible surface algorithm)
- Hidden-surface removal algorithms
  - object-space algorithm
    - Painter's algorithm
  - image-space algorithm
    - Z-buffer (depth buffer) algorithm



Z-buffer algorithm



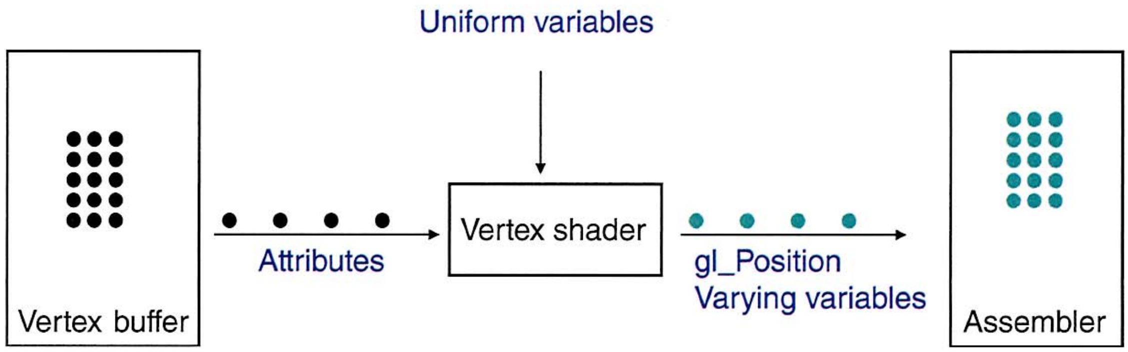


Color buffer

Z-buffer
(depth buffer)

```
glEnable (GL_DEPTH_TEST);
```

Vertex shader variables



- Three types of variables:
  - uniform: does not change per primitives; read-only in shaders
  - in (vertex sh.): input changes per vertex, read-only;
  - in (frag. sh.): interpolated input; read-only
  - out: shader-output; VS to FS; FS output.

Vertex shader (3D → … 2D)

```
uniform Matrix4 uModelViewMatrix;
uniform Matrix4 uNormalMatrix;
uniform Matrix4 uProjMatrix;

in vec3 aColor;
in vec4 aNormal;
in vec4 aVertex;

out vec3 vColor;
out vec3 vNormal;
out vec4 vPosition;
```

- Takes the object coordinates of every vertex position and turns them into eye coordinates, as well as the vertex's normal coordinates

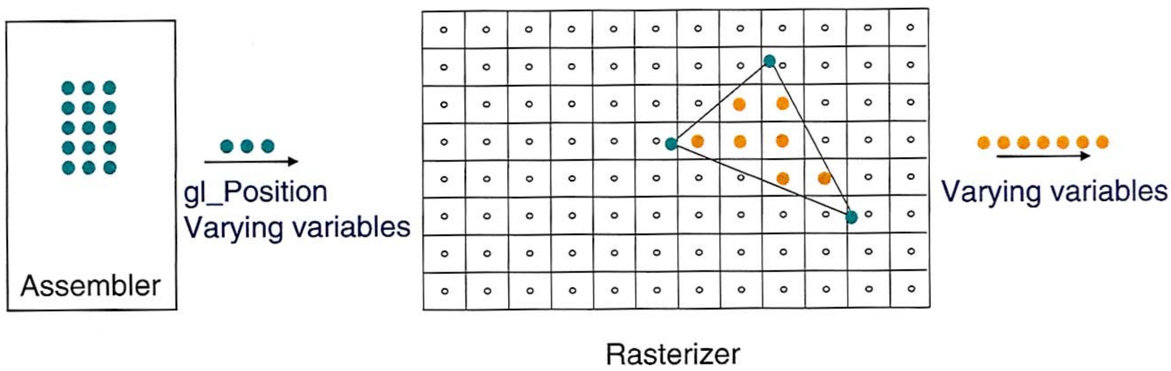
```
void main()
{
  vColor = aColor;
  vPosition = uModelViewMatrix * aVertex;
  vec4 normal = vec4(aNormal.x, aNormal.y, aNormal.z, 0.0);
  vNormal = vec3(uNormalMatrix * normal);
  gl_Position = uProjMatrix * vPosition;
}
```

$E^{-1}O\mathbf{c}$

$PE^{-1}O\mathbf{c}$

Clip coordinates

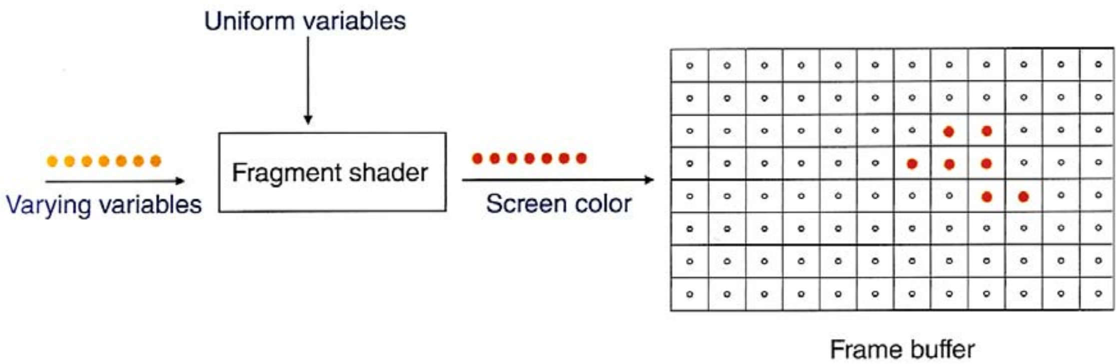
- Fixed function
  - Finds screen pixels inside of triangle
  - Interpolates values for the varying variables
  - vPosition at each pixel corresponds to geometric position of the point in the triangle observed at the pixel.



Fragment shader

- Compute material appearance
- By using uniform variables (light and color)
- We use the eye coordinates of variables (light, normal, position) in FS since rasterization was already done.

$E^{-1}O\mathbf{c}$



Fragment shader (pixel color)

- Simplest

```
in vec3 vColor;

out fragColor;

void main()
{
  fragColor = vec4(vColor.x, vColor.y, vColor.z, 1.0);
}
```


Fragment shader (pixel color)

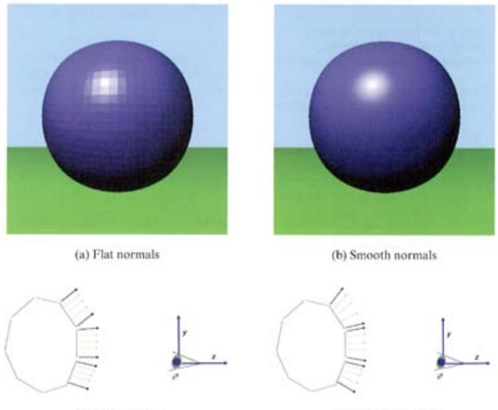
```
uniform vec3 uLight;
in vec3 vColor;
in vec3 vNormal;
in vec4 vPosition;

out fragColor;

void main()
{
  vec3 toLight = normalize(uLight - vec3(vPosition));
  vec3 normal = normalize(vNormal);
  float diffuse = max(0.0, dot(normal, toLight));
  vec3 intensity = vColor * diffuse;
  fragColor = vec4(intensity.x, intensity.y, intensity.z, 1.0);
}
```

Fragment shader (pixel color)

- Light computation

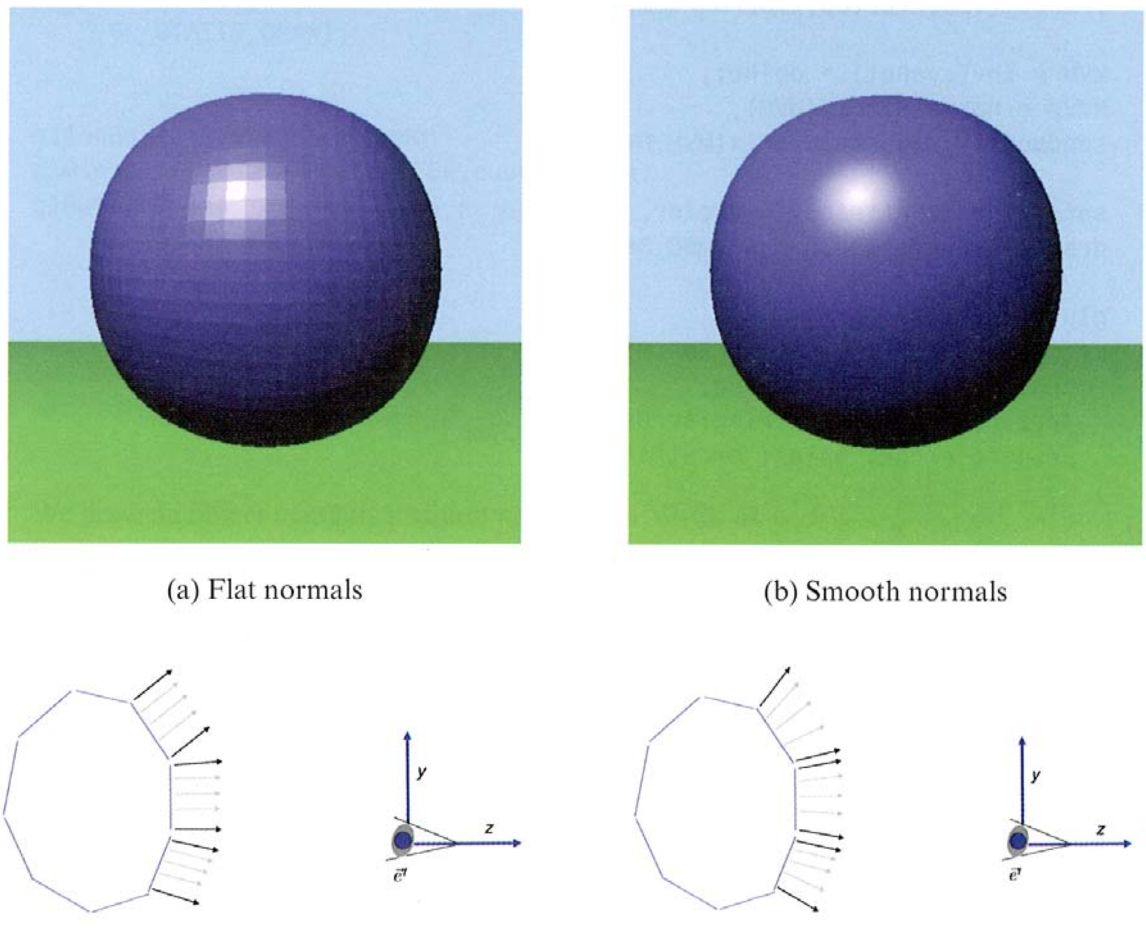


(a) Flat normals (b) Smooth normals (c) Flat normals (d) Smooth normals

```
uniform vec3 uLight;
in vec3 vColor;
in vec3 vNormal;
in vec4 vPosition;

out fragColor;

void main()
{
  vec3 toLight = normalize(uLight - vec3(vPosition));
  vec3 normal = normalize(vNormal);
  float diffuse = max(0.0, dot(normal, toLight));
  vec3 intensity = vColor * diffuse;
  fragColor = vec4(intensity.x, intensity.y, intensity.z, 1.0);
}
```

(a) Flat normals (b) Smooth normals

(c) Flat normals (d) Smooth normals

Detailed OpenGL & GLSL

- Also libraries we are using
- Lab Exercise!
- Consult the reference guides

- Please contact TA if you need any help!!

Homework #2

- Objectives
  - Understand 3D virtual environment
    - How they are represented
    - The process of making a picture from the 3D virtual world
  - Frames and Transformation
    - Object manipulation
    - View changes
    - 3D rotation
  - Interface