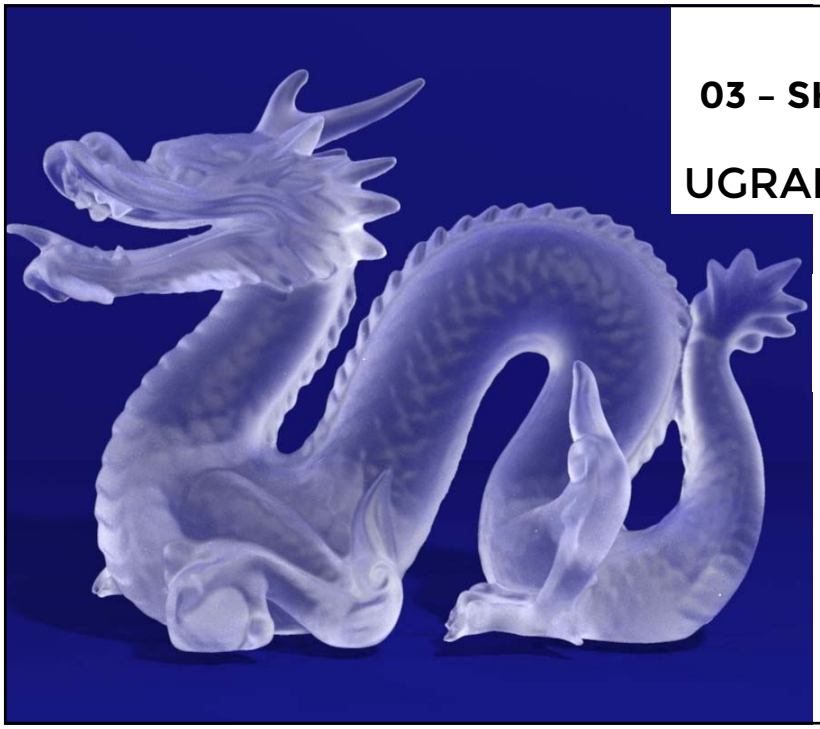


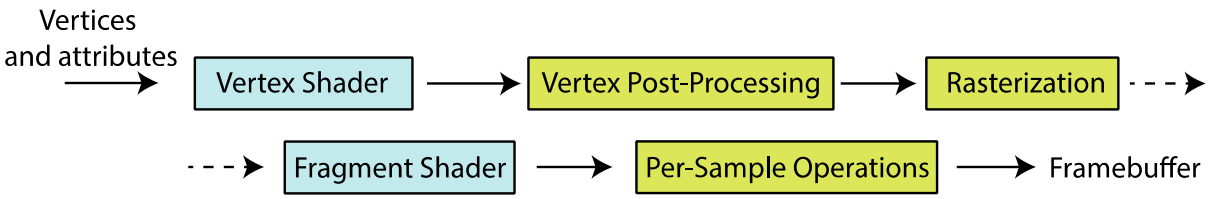
CPSC 314
03 – SHADERS, OPENGL, & JS

UGRAD.CS.UBC.CA/~CS314

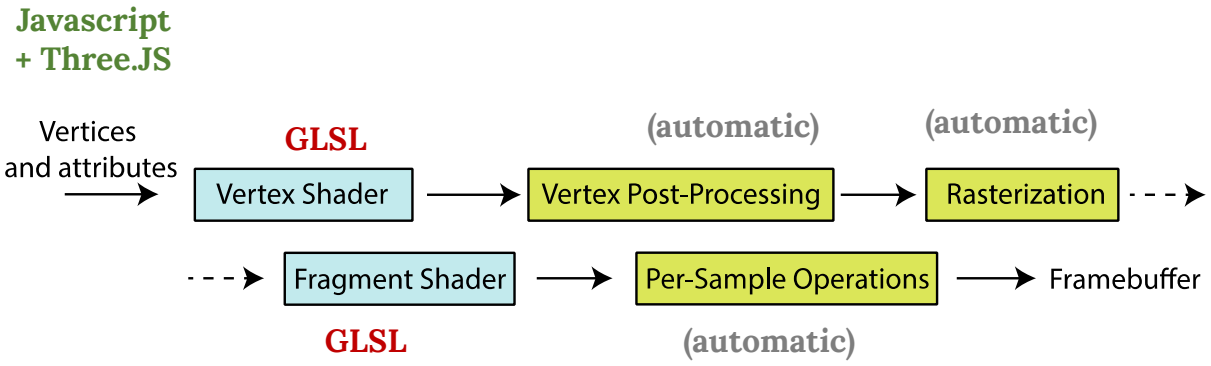
Textbook: Appendix A*
(helpful, but different version of OpenGL)

Alla Sheffer
Sep 2016

OPENGL RENDERING PIPELINE



OPENGL RENDERING PIPELINE



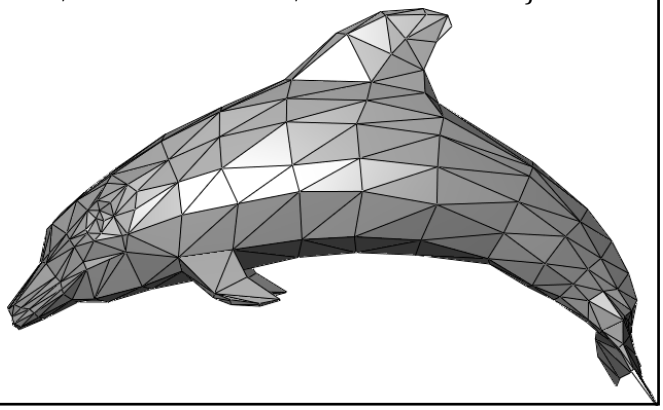
THREE.JS

- High-level library for Javascript
- Uses WebGL for rendering

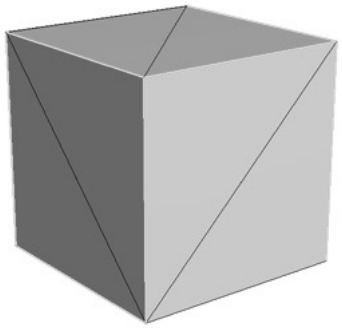
- Has **Scene**, **Mesh, Camera** objects
- **Scene** is hierarchical
- **Mesh** has geometry and material properties
- **Camera** is used for rendering

GEOMETRY

- Triangle meshes
  - Set of vertices
  - Triangle defines as {vertex_index1, vertex_index2, vertex_index3}

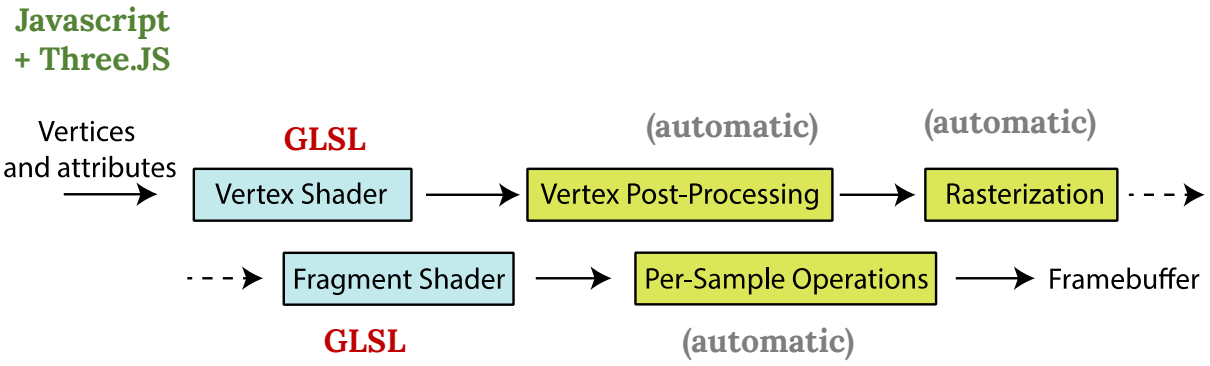


```
var verticesOfCube = [
    -1,-1,-1,    1,-1,-1,    1, 1,-1,    -1, 1,-1,
    -1,-1, 1,    1,-1, 1,    1, 1, 1,    -1, 1, 1,
];
var indicesOfFaces = [
    2,1,0,     0,3,2,
    0,4,7,     7,3,0,
    0,1,5,     5,4,0,
    1,2,6,     6,5,1,
    2,3,7,     7,6,2,
    4,5,6,     6,7,4
];
var geometry = new THREE.PolyhedronGeometry(
verticesOfCube, indicesOfFaces, 6, 2 );
```



GEOMETRY (JAVASCRIPT/THRE.JS)

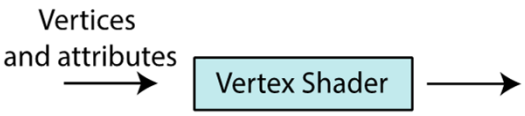
OPENGL RENDERING PIPELINE



GLSL

- OpenGL shading language
- Used for Fragment and Vertex shaders
- Lots of useful stuff:
  - vec3, vec4, dvec4, mat4, sampler2D
  - mat*vec, mat*mat
  - Reflect, refract
  - vec3 v(a.xy, 1)

VERTEX SHADER

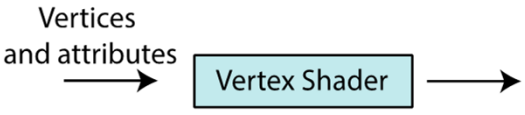


- VS is run for each vertex SEPARATELY
- By default doesn't know connectivity

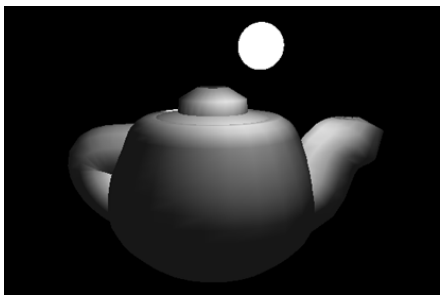
- Input: vertex coordinates in Object Coordinate System
- It's main goal is to set **gl_Position**

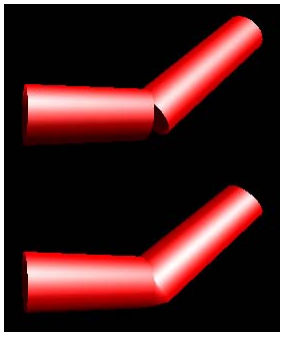
Object coordinates -> WORLD coordinates -> VIEW coordinates

VERTEX SHADER



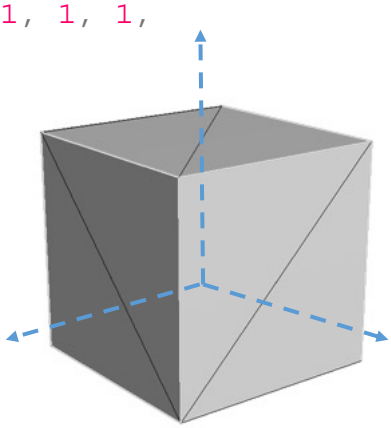
- Except simple conversion to world coordinates
- You can do anything with vertices (or anything that's passed)
  - e.g. deform vertices
  - e.g. skinning!





[courtesy NVIDIA]

```
var verticesOfCube = [
    -1,-1,-1,    1,-1,-1,    1, 1,-1,    -1, 1,-1,
    -1,-1, 1,    1,-1, 1,    1, 1, 1,    -1, 1, 1,
];
var indicesOfFaces = [
    2,1,0,    0,3,2,
    0,4,7,    7,3,0,
    0,1,5,    5,4,0,
    1,2,6,    6,5,1,
    2,3,7,    7,6,2,
    4,5,6,    6,7,4
];
var geometry = new THREE.PolyhedronGeometry(
verticesOfCube, indicesOfFaces, 6, 2 );
```



GEOMETRY (JAVASCRIPT/THRE.JS)

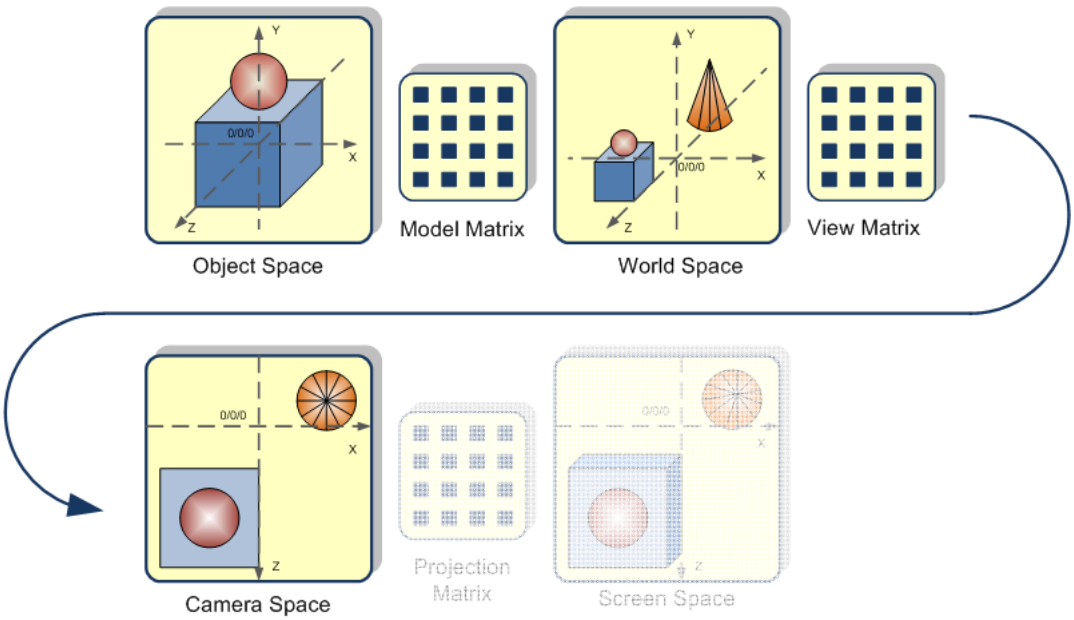


Image © Egon Rath

TRANSFORMATIONS SNEAK PEEK

• All the transformations are done via 4x4 matrices:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix}$$

TRANSFORMATIONS SNEAK PEEK

• All the transformations are done via 4x4 matrices:

Transformed Point Coordinates

Point Coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix}$$

Some weird matrix

TRANSFORMATIONS SNEAK PEEK

• What does this transformation do?

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = ?$$

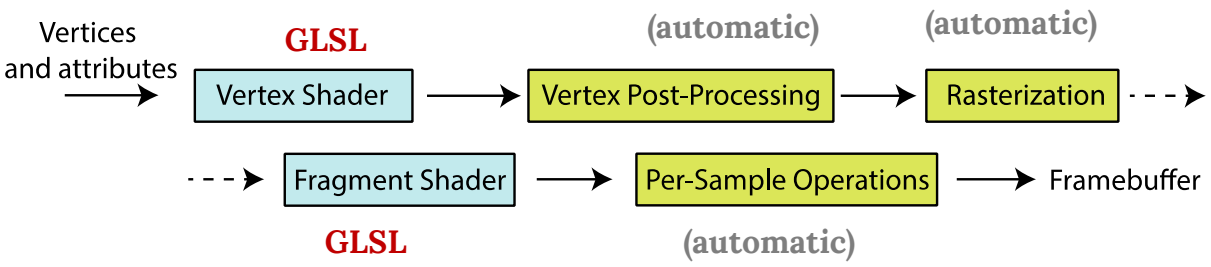
TRANSFORMATIONS SNEAK PEEK

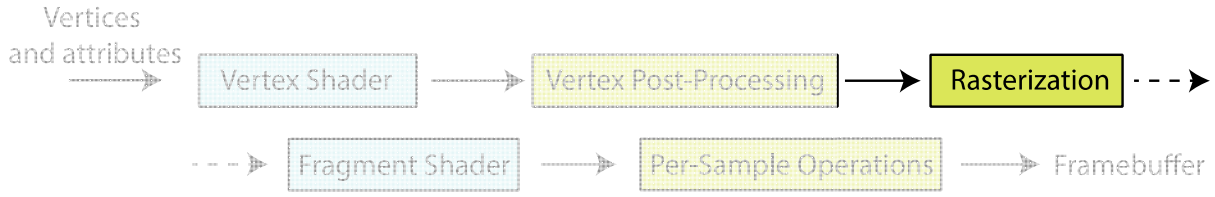
• What does this transformation do?

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1.0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1.0 \end{bmatrix} = \begin{bmatrix} x \\ 2y \\ 3z \\ 1.0 \end{bmatrix}$$

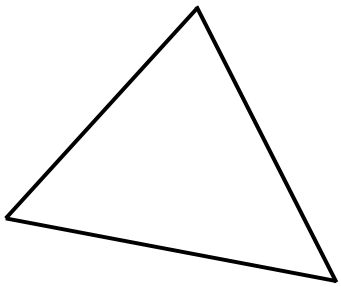
OPENGL RENDERING PIPELINE

Javascript
+ Three.JS

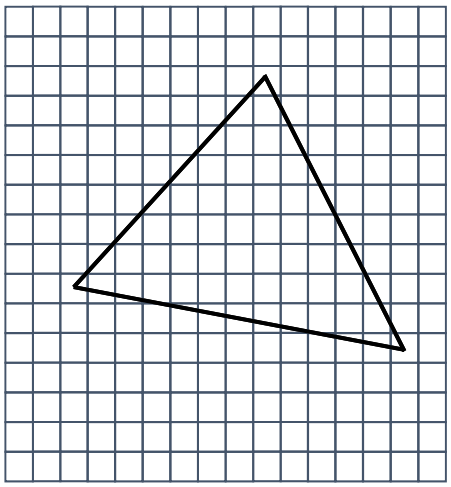




CAMERA VIEW

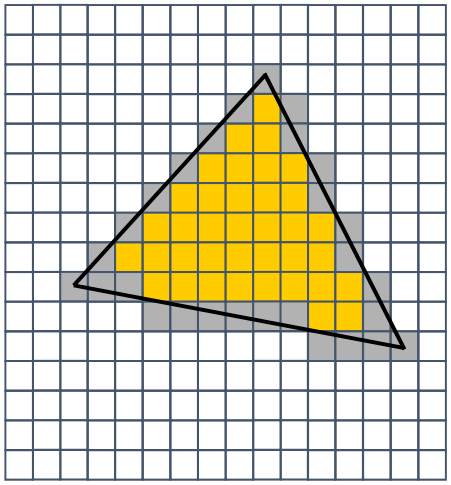


RASTERIZATION



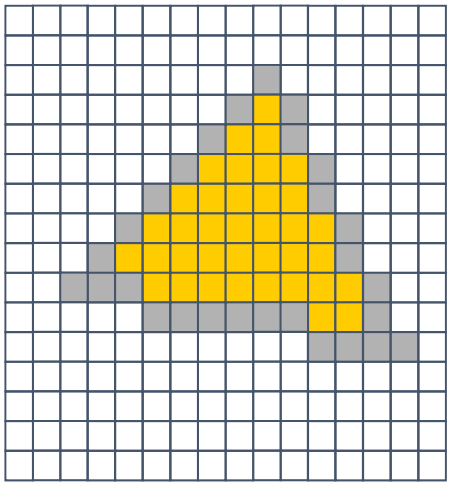
RASTERIZATION

•

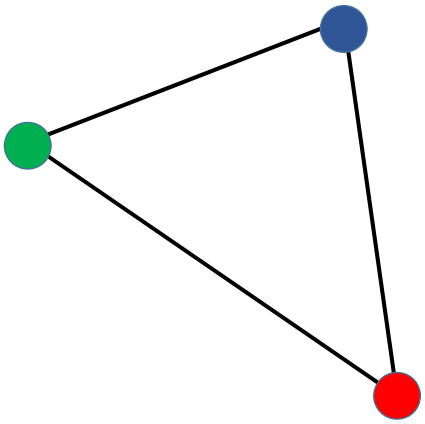


RASTERIZATION

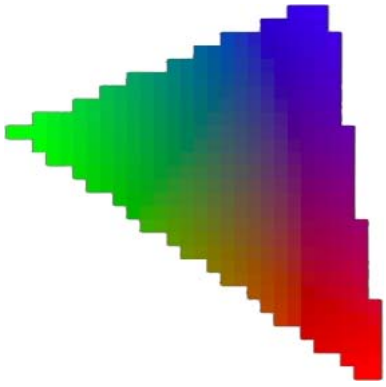
•



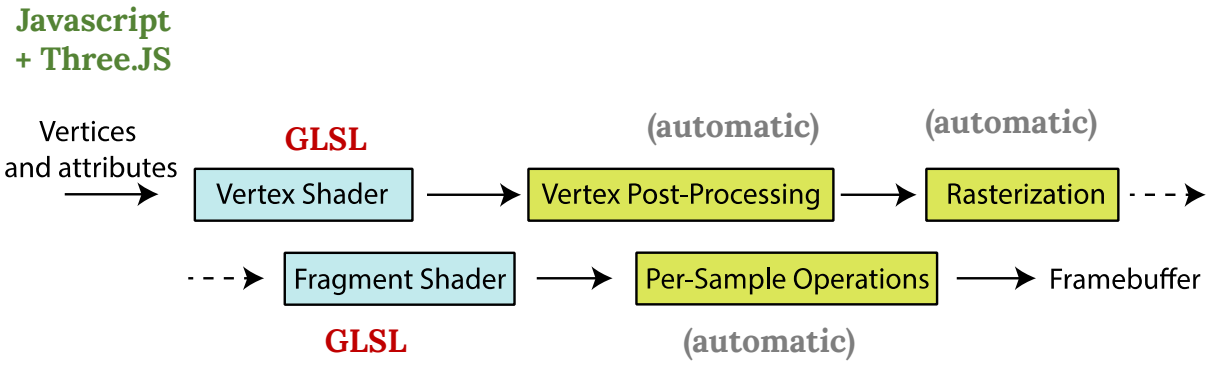
RASTERIZATION - INTERPOLATION



RASTERIZATION - INTERPOLATION

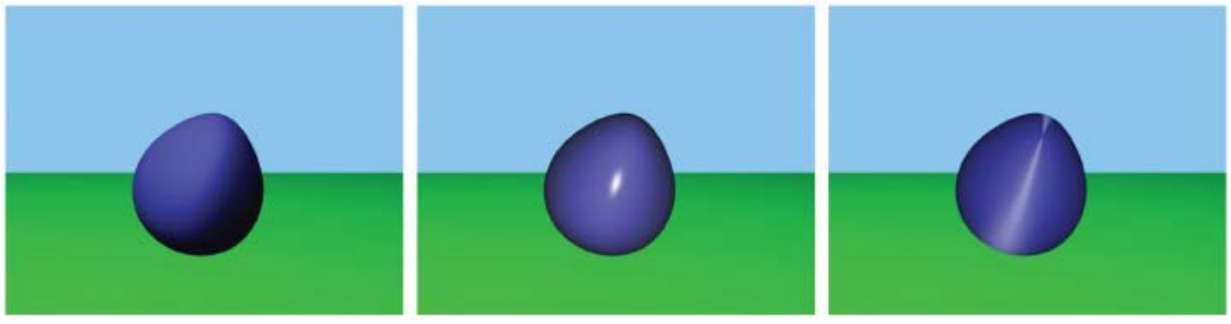


OPENGL RENDERING PIPELINE



FRAGMENT SHADER

- Fragment = data for drawing a pixel
- Has gl_FragCoord – 2D window coords
- May set color!



FRAGMENT SHADER

- Common Tasks:
  - texture mapping
  - per-pixel lighting and shading
- Synonymous with Pixel Shader

MINIMAL VERTEX SHADER

```
void main()
{
    // Transforming The Vertex
    gl_Position = modelViewMatrix * position;
}
```

MINIMAL FRAGMENT SHADER

```
void main()
{
    // Setting Each Pixel To Red
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

MINIMAL VERTEX SHADER

```
void main()
{
    // Transforming The Vertex
    gl_Position = modelViewMatrix * position;
}
```

defined by Three.JS

MINIMAL FRAGMENT SHADER

```
void main()
{
    // Setting Each Pixel To Red
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

MINIMAL VERTEX SHADER

```
void main()
{
    // Transforming The Vertex
    gl_Position = modelViewMatrix * position;
}
```

$$\begin{pmatrix} x \\ y \\ z \\ 1.0 \end{pmatrix}$$

defined by Three.JS

MINIMAL FRAGMENT SHADER

```
void main()
{
    // Setting Each Pixel To Red
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

MINIMAL VERTEX SHADER

```
void main()
{
    // Transforming The Vertex
    gl_Position = modelViewMatrix * position;
}
```

$\begin{pmatrix} x \\ y \\ z \\ 1.0 \end{pmatrix}$

view coordinate system

defined by Three.JS

MINIMAL FRAGMENT SHADER

```
void main()
{
    // Setting Each Pixel To Red
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```


MINIMAL VERTEX SHADER

```
void main()
{
    // Transforming The Vertex
    gl_Position = modelViewMatrix * position;
}
```

$\begin{pmatrix} x \\ y \\ z \\ 1.0 \end{pmatrix}$

view coordinate system

defined by Three.JS

MINIMAL FRAGMENT SHADER

```
void main()
{
    // Setting Each Pixel To Red
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```


Red Green Blue Alpha

VERTEX SHADER – EXAMPLE 2

```
uniform float uVertexScale;
attribute vec3 vColor;
varying vec3 fColor;

void main() {
 gl_Position = vec4(position.x * uVertexScale, position.y, 0.0,1.0);
 fColor = vColor;
}
```

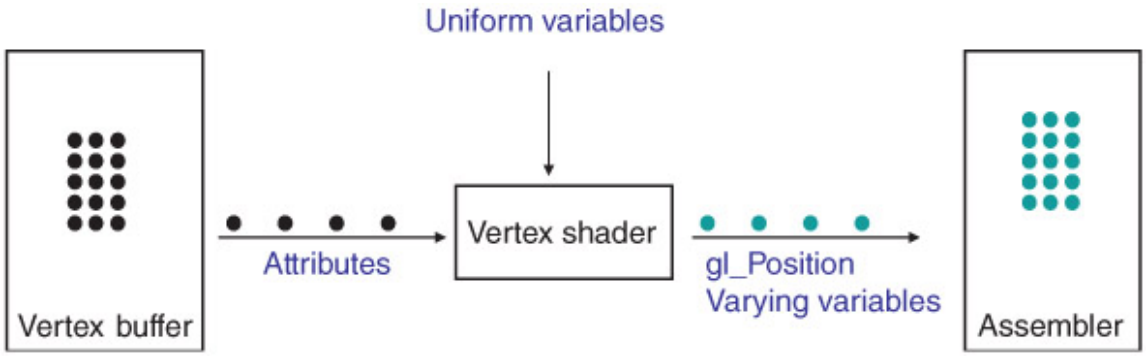
CONCEPTS

- uniform
  - same for all vertices
- varying
  - computed per vertex, automatically interpolated for fragments
- attribute
  - some values per vertex
  - available only in Vertex Shader

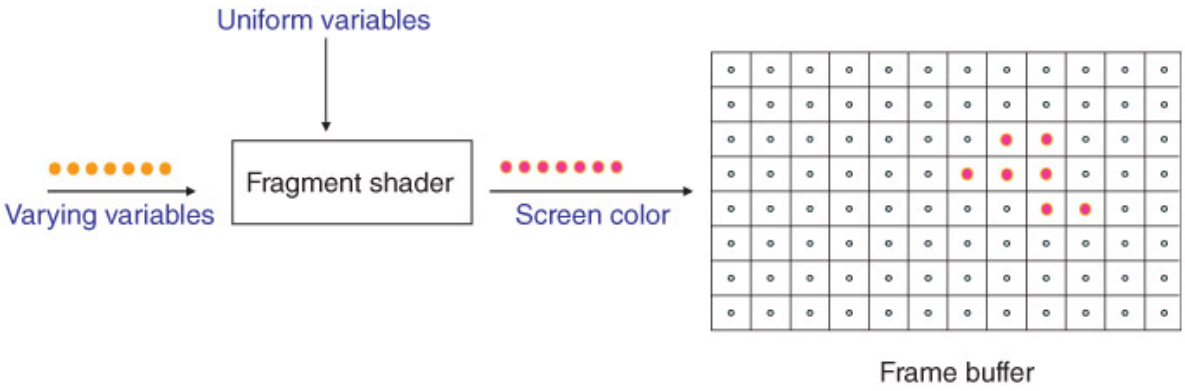
CONCEPTS

- uniform JS + Three.JS → Vertex Shader → Fragment Shader
  - same for all vertices
- varying Vertex Shader → Fragment Shader
  - computed per vertex, automatically interpolated for fragments
- attribute JS + Three.JS → Vertex Shader
  - some values per vertex
  - available only in Vertex Shader

VERTEX SHADER



FRAGMENT SHADER



ATTACHING SHADERS

```
var remoteMaterial = new THREE.ShaderMaterial({
  uniforms: {
  remotePosition: remotePosition,
 },});
//here goes loading shader files into shaders[] …
remoteMaterial.vertexShader = shaders['glsl/remote.vs.glsl'];
remoteMaterial.fragmentShader = shaders['glsl/remote.fs'];
var remoteGeometry = new THREE.SphereGeometry(1, 32, 32);
var remote = new THREE.Mesh(remoteGeometry, remoteMaterial);

scene.add(remote);
```