Introduction to OpenGL ES 2.0 Programming

folkert@feedface.com

This Talk

1) OpenGL ES Concepts
2) Linear Algebra (just a tiny bit)
3) hello-camp.go

https://www.feedface.com/tech/intro-opengles.html

NOT This Talk

Vendor Extensions

Clipping

Geometry Shaders

Stencils

Lighting

Framebuffer Objects

Framerate

Culling

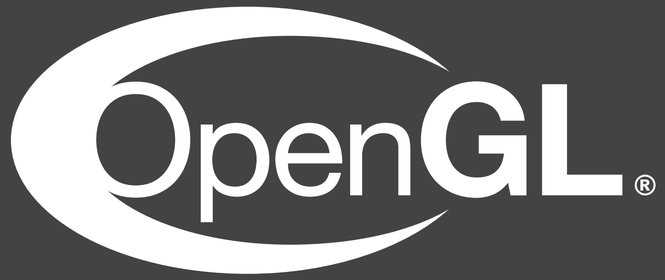
Tesselation Shaders

Quaternions

Shading

Blending

OpenGL ES



- API for rendering 2D and 3D Computer Graphics
- Developed by SGI in 1992
- Open Standard, now maintained by Khronos Group consortium
- Used in CAD, Visualization, Desktop Compositing, Art (Demo Scene!), Games, VR, AR ...

- OpenGL for Embedded Systems
- Implementations by NVIDIA, AMD, Intel, Arm, Apple, Samsung, Broadcom, Nintendo, Sony ...
- Available on Linux, MacOS, Windows, iOS, Android ...
- Bindings for C, C++, Java, C#, Python, Ruby, Go ...
- Latest version OpenGL ES 3.2
    - but anything from OpenGL ES 2.0 onwards is cool!

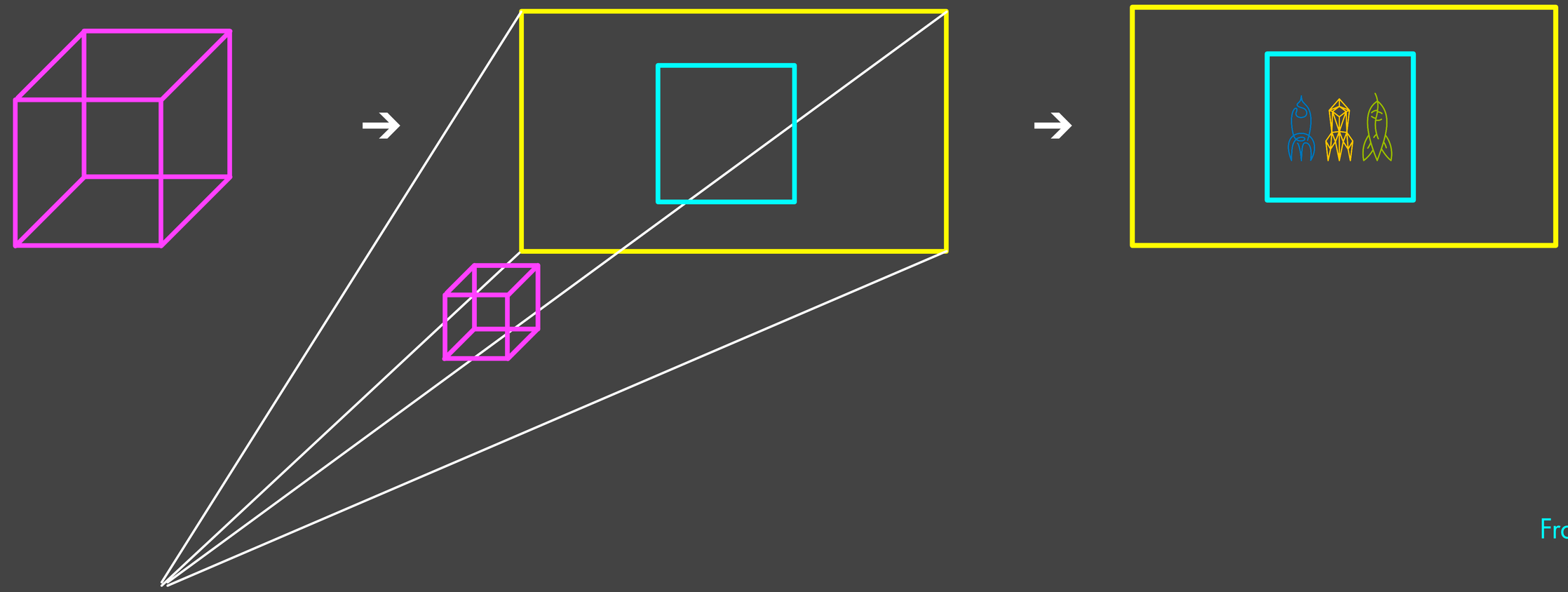
- Native Platform Interface
- Creates & Configures OpenGL ES Context (view/window/fullscreen)
- Your platform might come with something else (Mesa, WGL, CGL, GLX, ...)

Rasterizer

Geometry

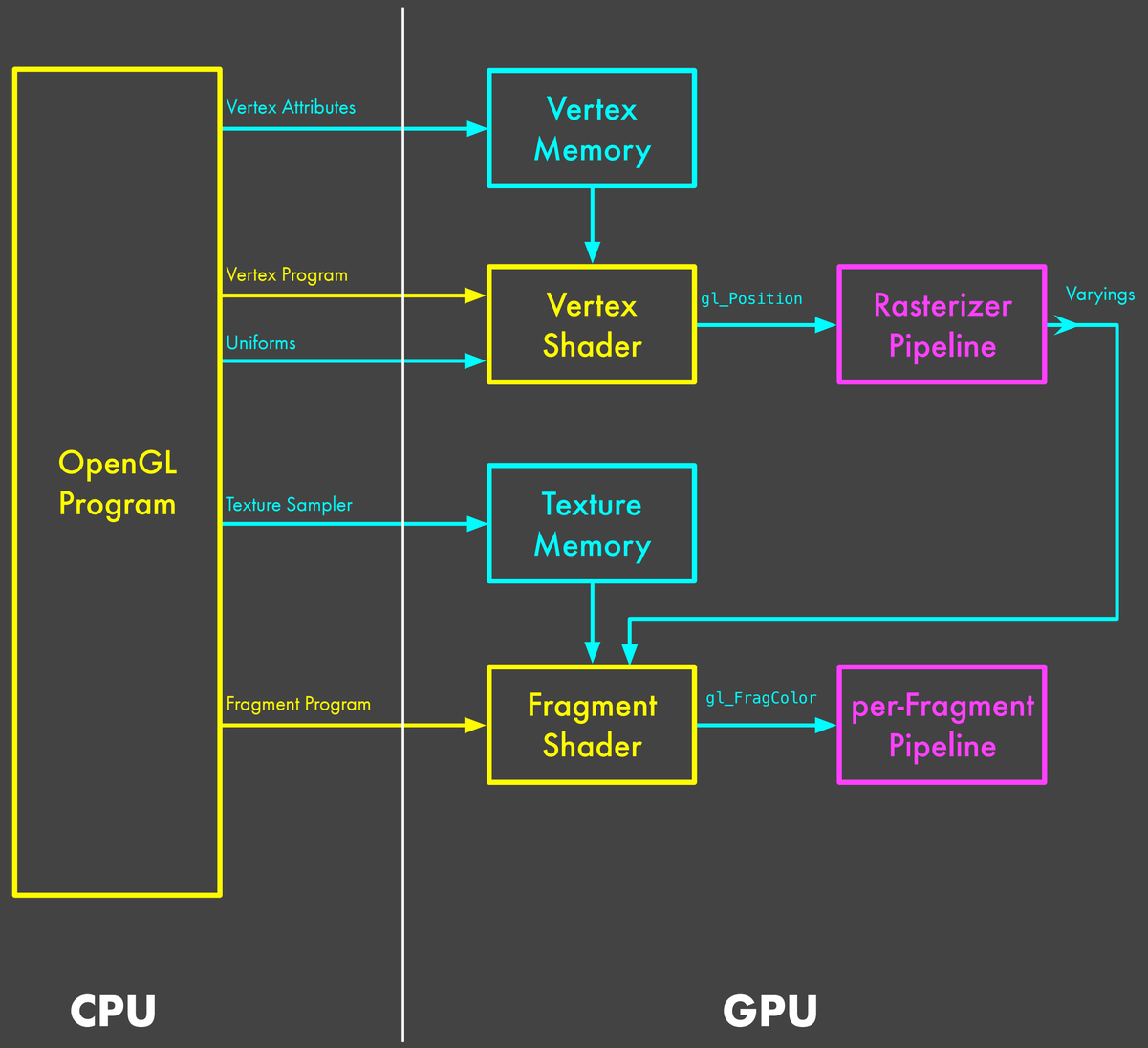
Projection

Texture Mapping



Vertex
Fragment
Screen

Rendering Architecture



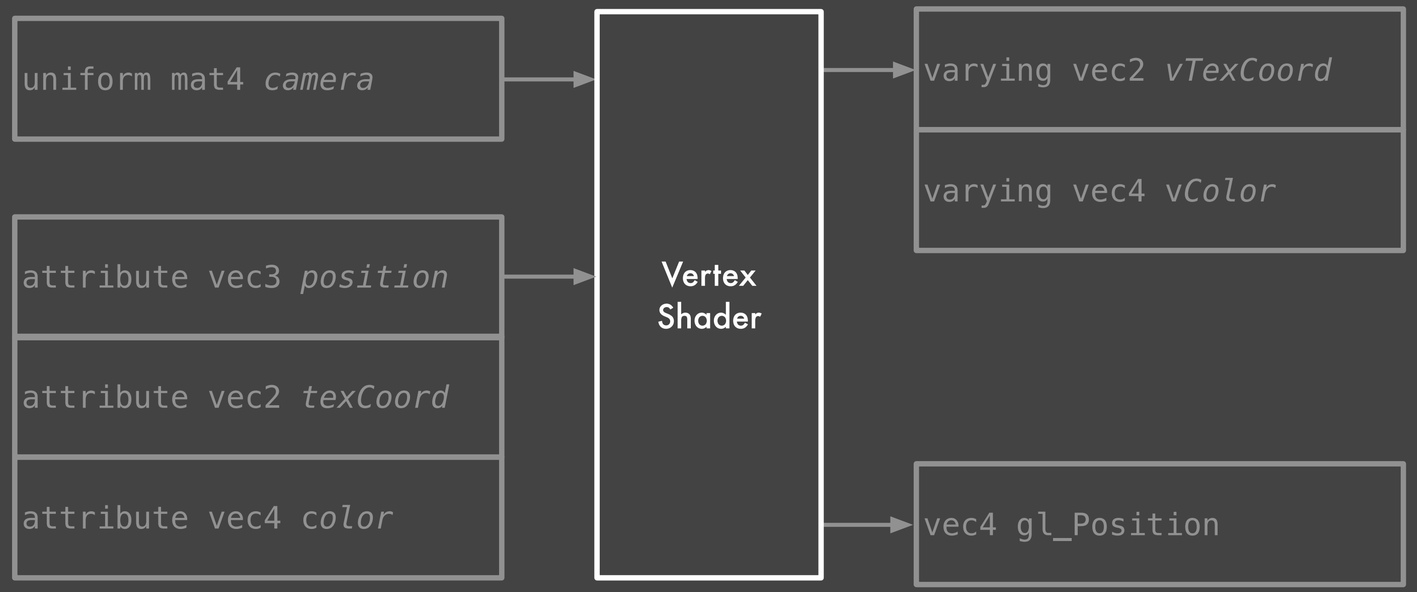
Code
Data
Metal

GLSL Shader Language

- C-like Syntax
- Vector and Matrix types (float, vec2, vec3, vec4, mat4 ...)
- Built-in math functions (sin, cos, mod, clamp, sqrt, exp, log, dot, cross...)
- No recursion
- Limited Loops

Vertex Shader

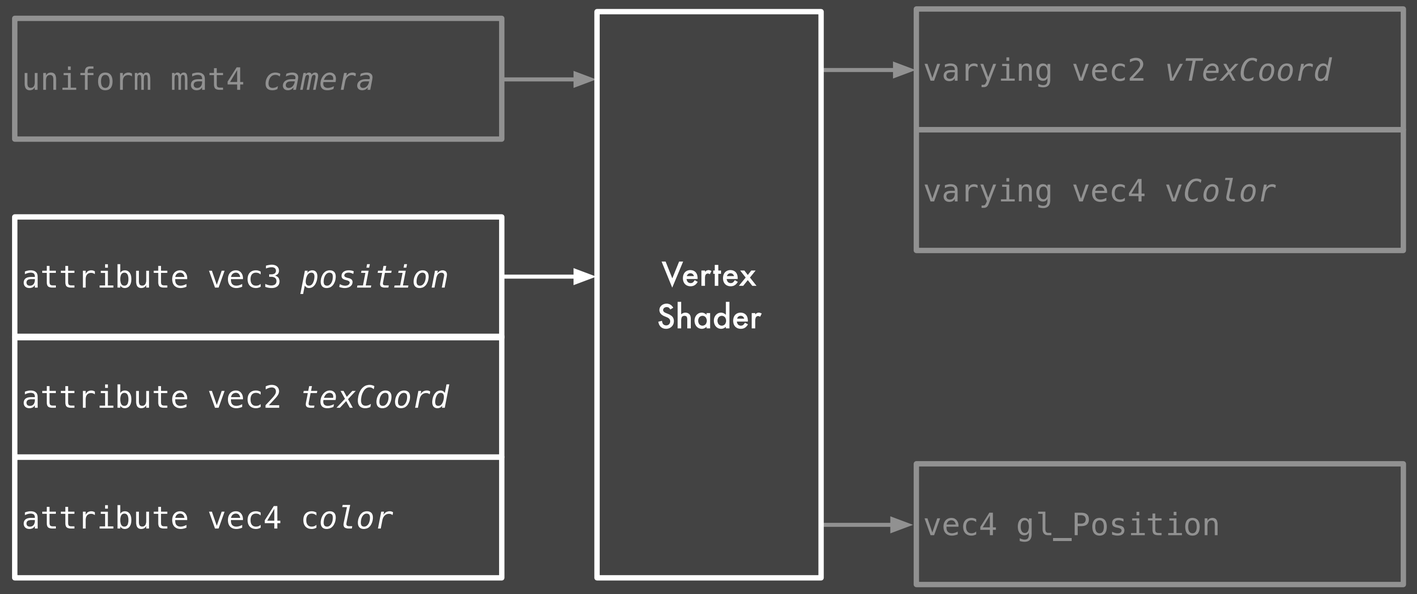
- Runs once for each vertex of each triangle of the geometry
- Get per-vertex input from `attribute`
- Get parameters from `uniform`
- Pass output to fragment shader in `varying`
- Pass position vector to rasterizer pipeline in `gl_Position`




```
uniform mat4 camera;
attribute vec3 position;
attribute vec2 texCoord; varying vec2 vTexCoord;
attribute vec4 color;    varying vec4 vColor;

void main() {
    vColor = color;
    vTexCoord = texCoord;
    gl_Position = camera * vec4(position, 1);
}
```

Vertex Shader

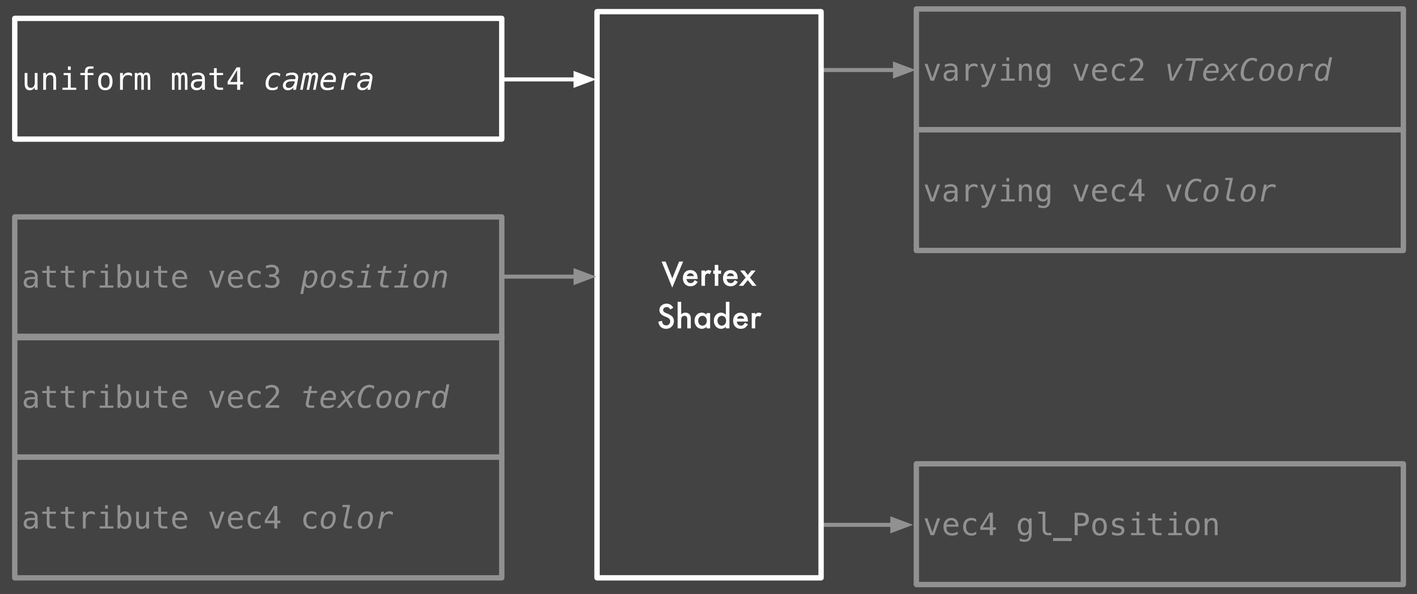
- Runs once for each vertex of each triangle of the geometry
- Get per-vertex input from `attribute`
- Get parameters from `uniform`
- Pass output to fragment shader in `varying`
- Pass position vector to rasterizer pipeline in `gl_Position`




```
uniform mat4 camera;
attribute vec3 position;
attribute vec2 texCoord;  varying vec2 vTexCoord;
attribute vec4 color;     varying vec4 vColor;

void main() {
    vColor = color;
    vTexCoord = texCoord;
    gl_Position = camera * vec4(position, 1);
}
```

Vertex Shader

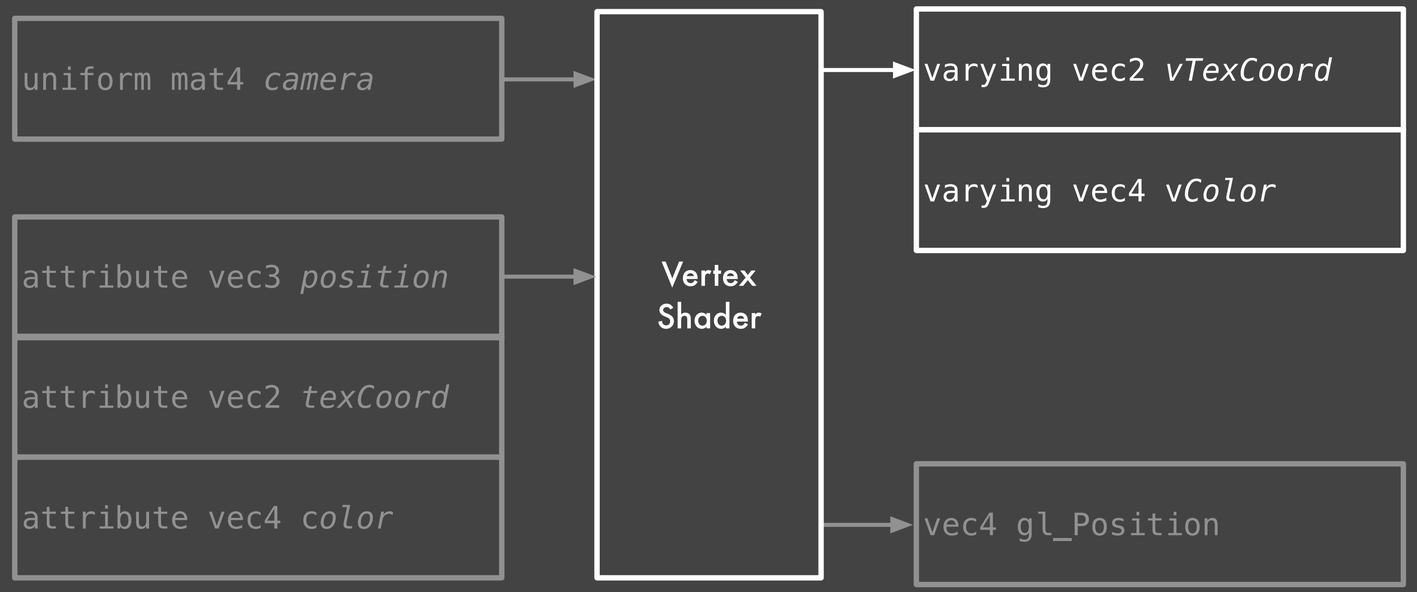
- Runs once for each vertex of each triangle of the geometry
- Get per-vertex input from `attribute`
- Get parameters from `uniform`
- Pass output to fragment shader in `varying`
- Pass position vector to rasterizer pipeline in `gl_Position`




```
uniform mat4 camera;
attribute vec3 position;
attribute vec2 texCoord; varying vec2 vTexCoord;
attribute vec4 color;    varying vec4 vColor;

void main() {
    vColor = color;
    vTexCoord = texCoord;
    gl_Position = camera * vec4(position, 1);
}
```

Vertex Shader

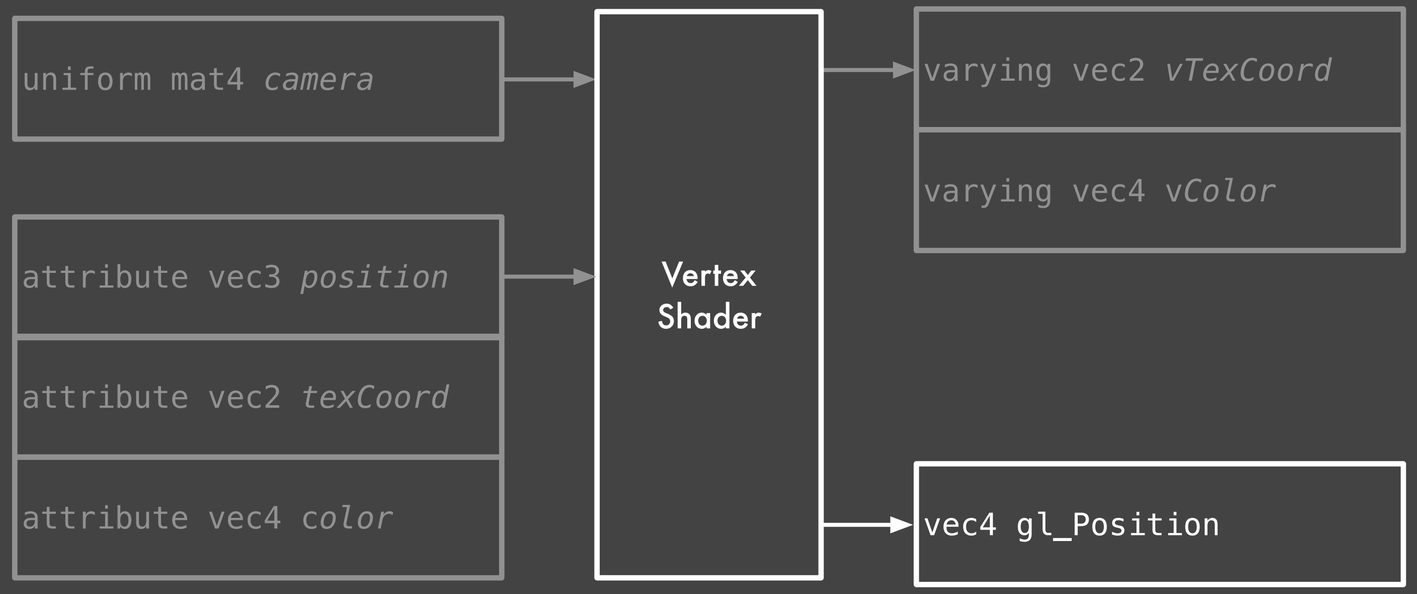
- Runs once for each vertex of each triangle of the geometry
- Get per-vertex input from `attribute`
- Get parameters from `uniform`
- Pass output to fragment shader in `varying`
- Pass position vector to rasterizer pipeline in `gl_Position`




```
uniform mat4 camera;
attribute vec3 position;
attribute vec2 texCoord;  varying vec2 vTexCoord;
attribute vec4 color;     varying vec4 vColor;

void main() {
    vColor = color;
    vTexCoord = texCoord;
    gl_Position = camera * vec4(position, 1);
}
```

Vertex Shader

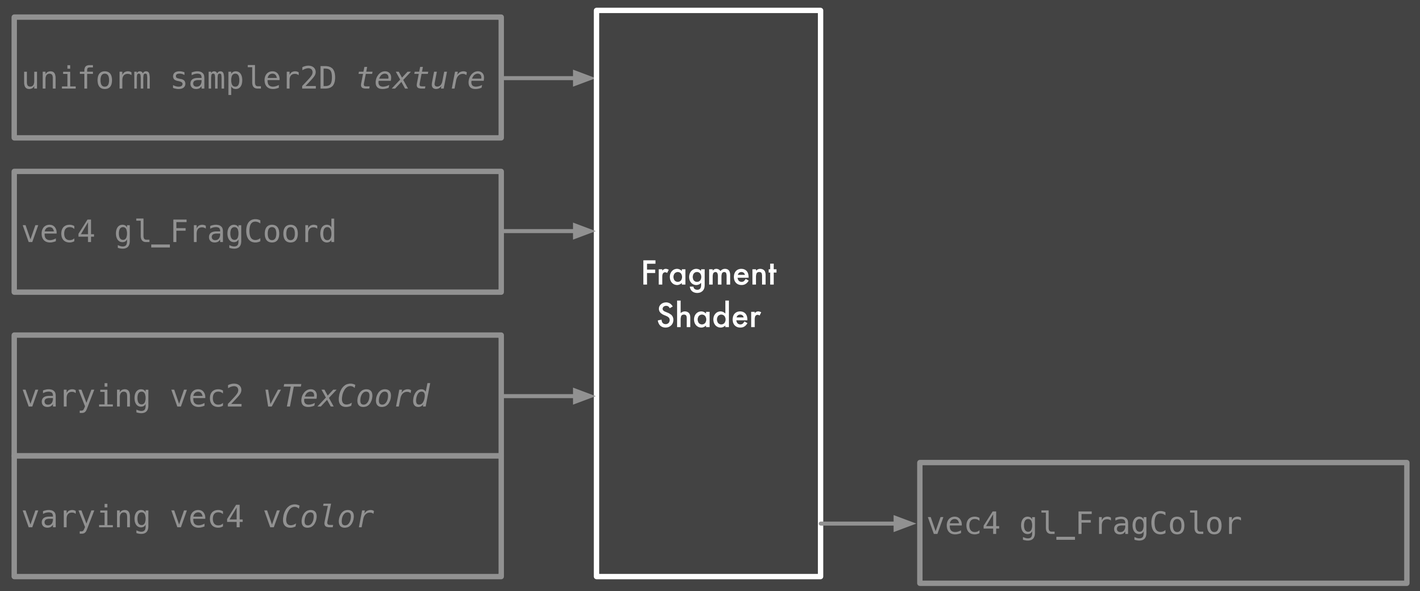
- Runs once for each vertex of each triangle of the geometry
- Get per-vertex input from `attribute`
- Get parameters from `uniform`
- Pass output to fragment shader in `varying`
- Pass position vector to rasterizer pipeline in `gl_Position`




```
uniform mat4 camera;
attribute vec3 position;
attribute vec2 texCoord; varying vec2 vTexCoord;
attribute vec4 color;    varying vec4 vColor;

void main() {
    vColor = color;
    vTexCoord = texCoord;
    gl_Position = camera * vec4(position, 1);
}
```

Fragment Shader

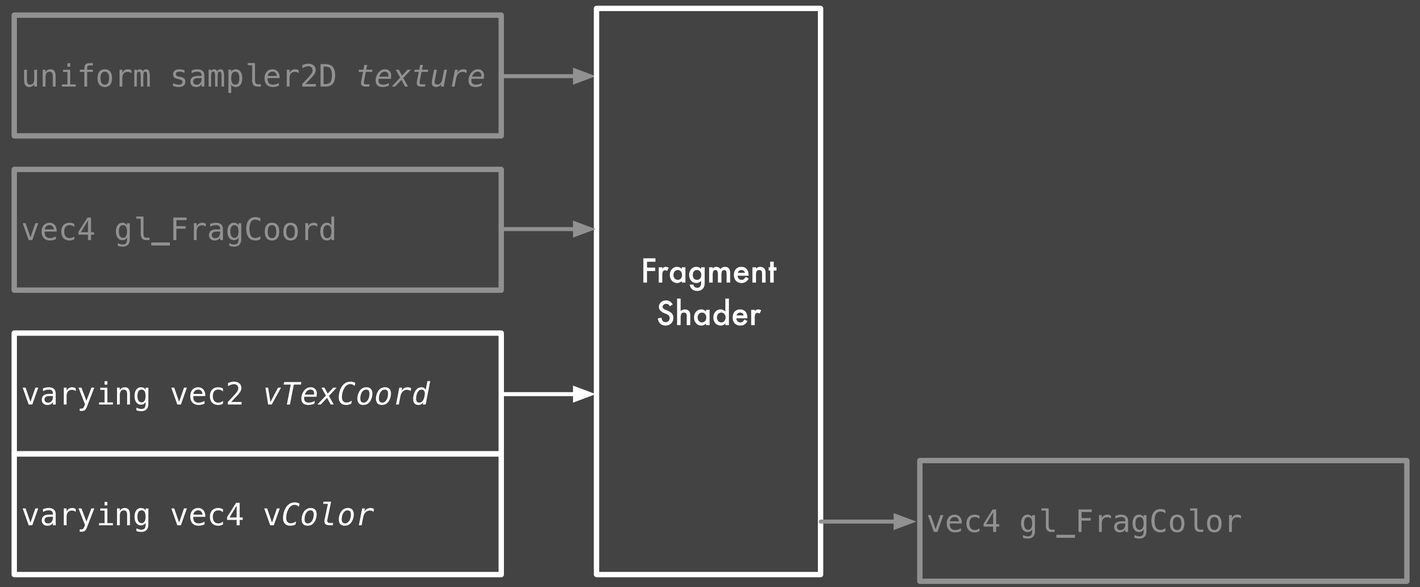
- Runs once for each fragment of each rasterized triangle
- Get per-vertex input from `varying`
- Get parameters from `uniform`
- Get texture data from `uniform sampler2D`
- Pass color vector to fragment pipeline in `gl_FragColor`




```
uniform sampler2D texture;
varying vec2 vTexCoord;
varying vec4 vColor;

void main() {
    vec4 texColor = texture2D(texture,vTexCoord);
    gl_FragColor = vec4( vColor.rgb, 1. - texColor.a );
}
```

Fragment Shader

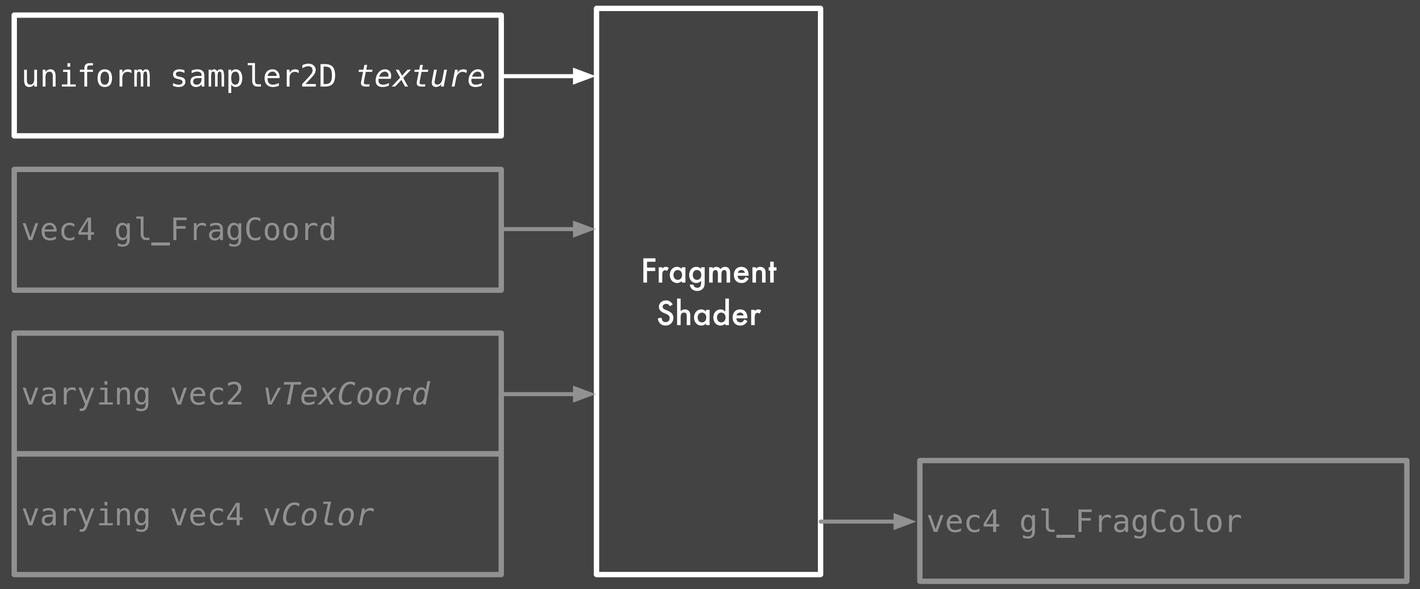
- Runs once for each fragment of each rasterized triangle
- Get per-vertex input from `varying`
- Get parameters from `uniform`
- Get texture data from `uniform sampler2D`
- Pass color vector to fragment pipeline in `gl_FragColor`




```
uniform sampler2D texture;
varying vec2 vTexCoord;
varying vec4 vColor;

void main() {
    vec4 texColor = texture2D(texture,vTexCoord);
    gl_FragColor = vec4( vColor.rgb, 1. - texColor.a );
}
```

Fragment Shader

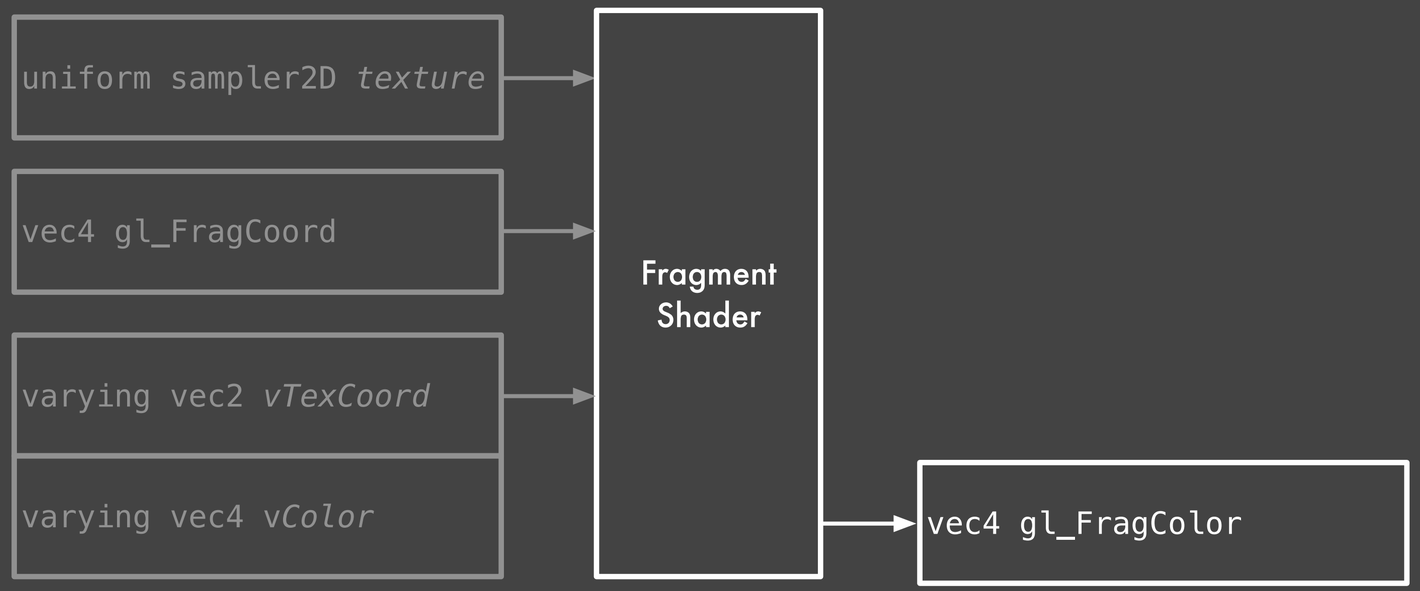
- Runs once for each fragment of each rasterized triangle
- Get per-vertex input from `varying`
- Get parameters from `uniform`
- Get texture data from `uniform sampler2D`
- Pass color vector to fragment pipeline in `gl_FragColor`




```
uniform sampler2D texture;
varying vec2 vTexCoord;
varying vec4 vColor;

void main() {
    vec4 texColor = texture2D(texture,vTexCoord);
    gl_FragColor = vec4( vColor.rgb, 1. - texColor.a );
}
```

Fragment Shader

- Runs once for each fragment of each rasterized triangle
- Get per-vertex input from `varying`
- Get parameters from `uniform`
- Get texture data from `uniform sampler2D`
- Pass color vector to fragment pipeline in `gl_FragColor`



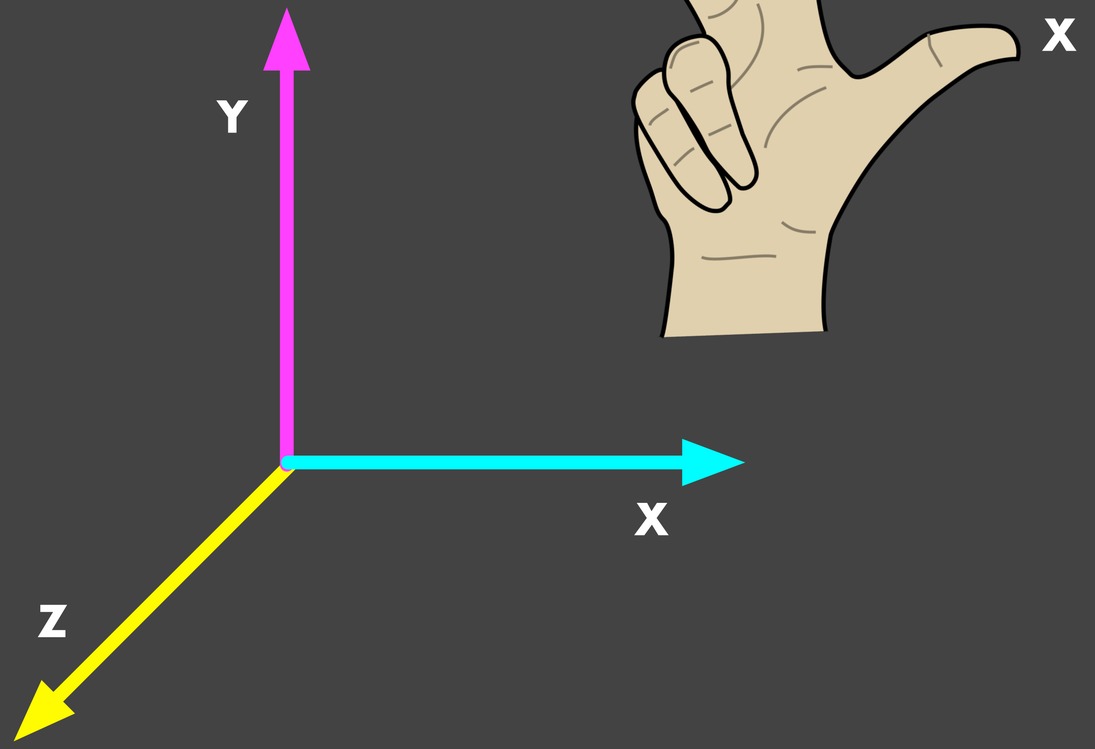

```
uniform sampler2D texture;
varying vec2 vTexCoord;
varying vec4 vColor;

void main() {
    vec4 texColor = texture2D(texture,vTexCoord);
    gl_FragColor = vec4( vColor.rgb, 1. - texColor.a );
}
```

thats the basics!

(a tiny bit of)
Linear Algebra

World Coordinates

- Right-Hand Coordinate System
- X Axis points RIGHT
- Y Axis points UP
- Z Axis points AT YOU



Vectors and Matrices

- Vectors describe coordinates

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Matrices describe transformations

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Vectors

- describe vertex positions in space
- or texture coordinates on a surface
- or colors
- or other things

```
vec3 p = vec3(x,y,z);
```

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

```
vec2 t = vec2(x,y);
```

$$\vec{t} = \begin{pmatrix} x \\ y \end{pmatrix}$$

```
vec4 c = vec4(r,g,b,a);
```

$$\vec{c} = \begin{pmatrix} r \\ g \\ b \\ a \end{pmatrix}$$

Vector Operations

- Scalar Multiplication ('dot product')
  - Multiplication of a vector with itself yields the length of the vector.
  - Multiplication of two vectors yields the cosine of the angle between the vectors
- Vector Multiplication ('cross product')
  - Multiplication of two vectors yields a vector that is orthogonal to the two vectors.

```
float l = dot(v,v);
```

$$l = \vec{v} \cdot \vec{v}$$

```
float a = acos( dot(u,v) );
```

$$\alpha = \text{acos}\left(\vec{u} \cdot \vec{v}\right)$$

```
vec3 n = cross(u,v);
```

$$\vec{n} = \vec{u} \times \vec{v}$$

Vector Operations

- Scalar Multiplication ('dot product')
  - Multiplication of a vector with itself yields the length of the vector.
  - Multiplication of two vectors yields the cosine of the angle between the vectors
- Vector Multiplication ('cross product')
  - Multiplication of two vectors yields a vector that is orthogonal to the two vectors.

```
float l = dot(v,v);
```

$$l = \vec{v} \cdot \vec{v}$$

```
float a = acos( dot(u,v) );
```

$$\alpha = \text{acos}\left(\vec{u} \cdot \vec{v}\right)$$

```
vec3 n = cross(u,v);
```

$$\vec{n} = \vec{u} \times \vec{v}$$

Vector Operations

- Scalar Multiplication ('dot product')
  - Multiplication of a vector with itself yields the length of the vector.
  - Multiplication of two vectors yields the cosine of the angle between the vectors
- Vector Multiplication ('cross product')
  - Multiplication of two vectors yields a vector that is orthogonal to the two vectors.

```
float l = dot(v,v);
```

$$l = \vec{v} \cdot \vec{v}$$

```
float a = acos( dot(u,v) );
```

$$\alpha = \text{acos}\left(\vec{u} \cdot \vec{v}\right)$$

```
vec3 n = cross(u,v);
```

$$\vec{n} = \vec{u} \times \vec{v}$$

Matrices

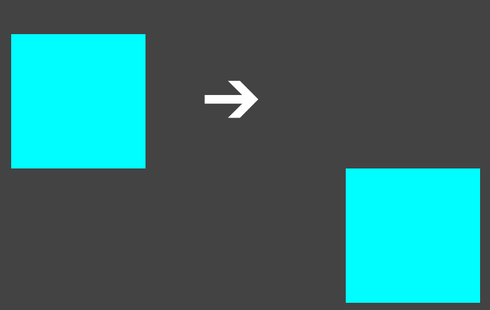
- describe transformations in R_3 space
- have dimensions 4×4

```
mat4 M = mat4();
```

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

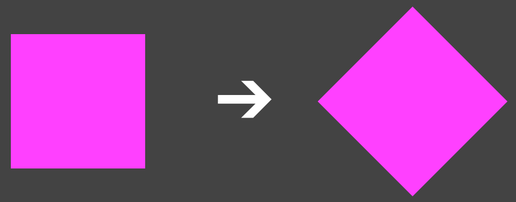
Transformation Matrices

Translate



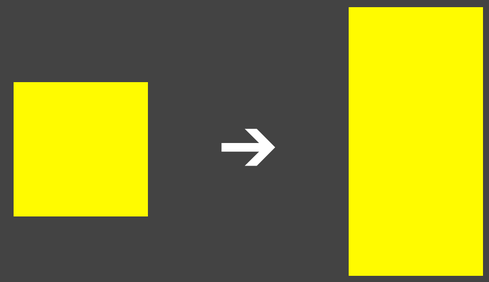
$$T = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate



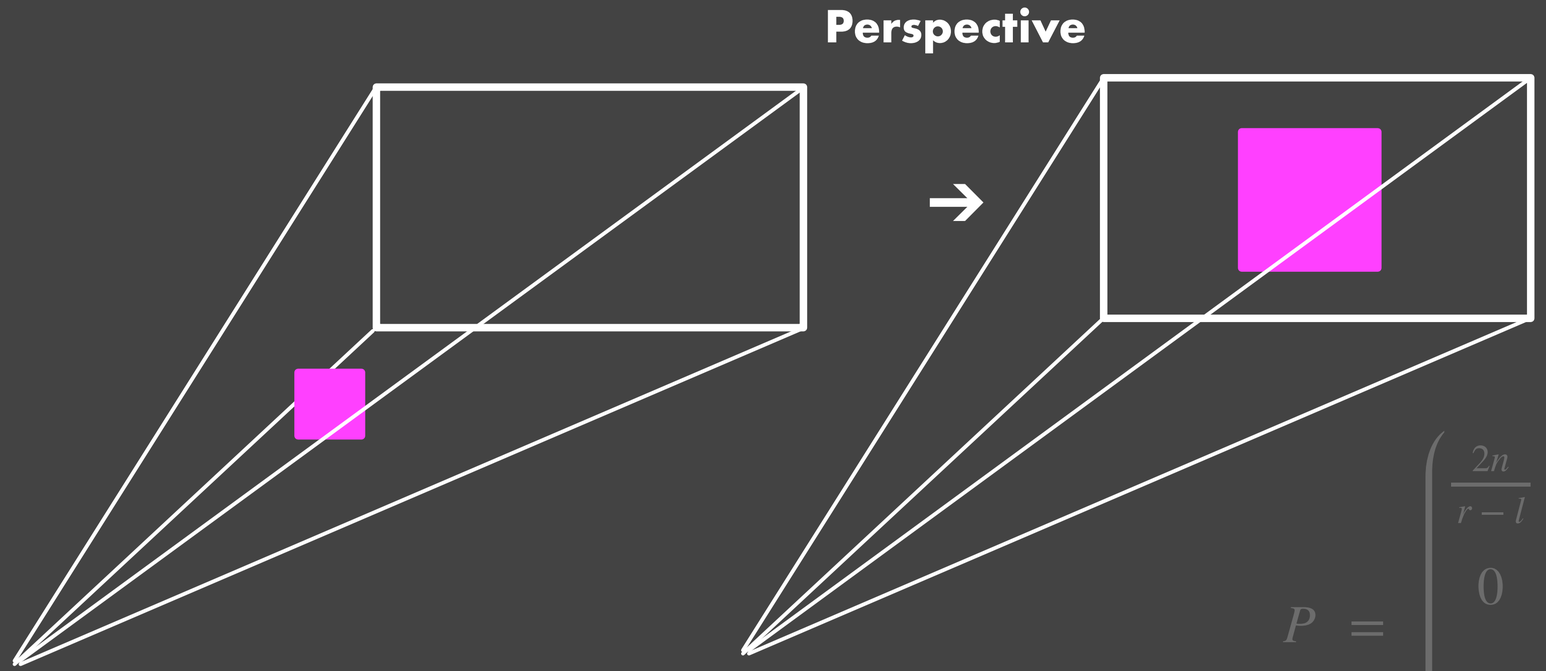
$$R_z = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scale



$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Projection Matrix



$$P = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{2\text{fn}}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Matrix Operations

- Multiplication of a vector with a matrix yields a vector with the matrix transformation applied.

```
vec4 v1 = M * v;
```

$$\vec{v}' = M\vec{v}$$

- Multiplication of two matrices yields a matrix that describes the two matrix transformation chained.

```
mat4 M0 = M * N;
```

$$M' = MN$$

Matrix Operations

- Multiplication of a vector with a matrix yields a vector with the matrix transformation applied.

```
vec4 v0 = M * v;
```

$$\vec{v}' = M\vec{v}$$

- Multiplication of two matrices yields a matrix that describes the two matrix transformation chained.

```
mat4 M1 = M * N;
```

$$M' = MN$$

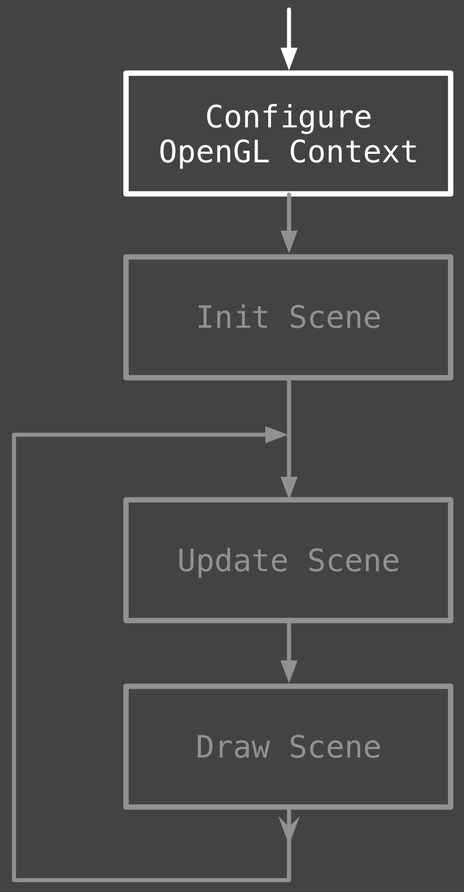
enough algebra for now :)

hello-camp.go

https://www.feedface.com/tech/hello-camp.go

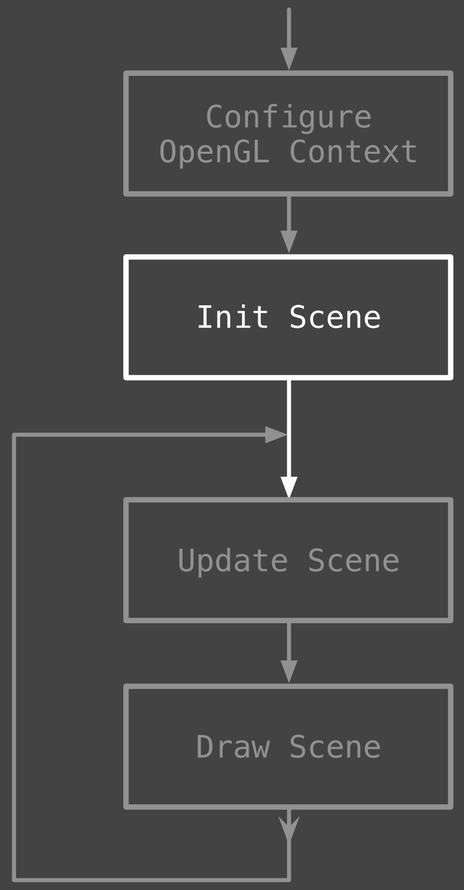
Program Flow

- Configure OpenGL Context
- Initialize Scene:
  - Init Vertex Data
  - Init Texture Data
  - Init Shaders
  - Init Shader Program
  - Init Camera Matrix
- Enter Render Loop:
  - Update Scene
  - Draw Scene
  - Repeat



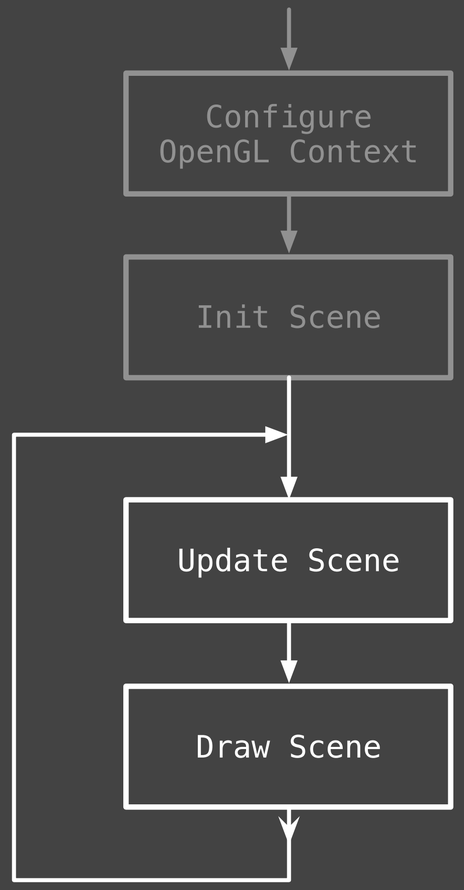
Program Flow

- Configure OpenGL Context
- Initialize Scene:
  - Init Vertex Data
  - Init Texture Data
  - Init Shaders
  - Init Shader Program
  - Init Camera Matrix
- Enter Render Loop:
  - Update Scene
  - Draw Scene
  - Repeat



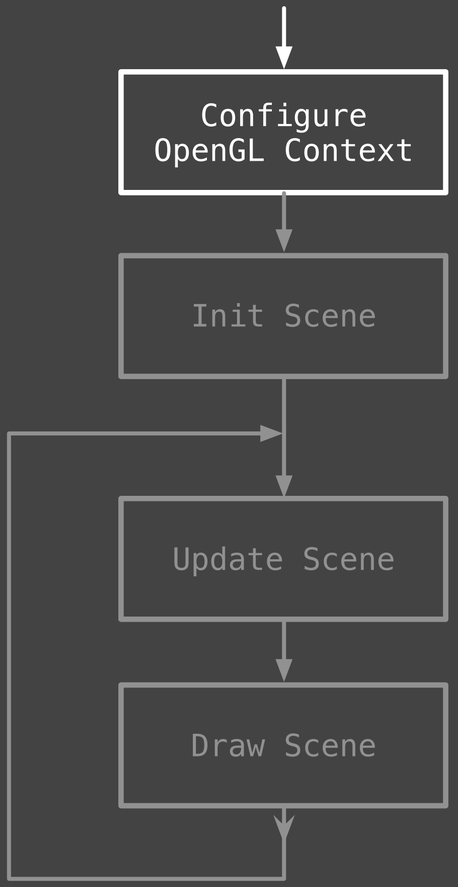
Program Flow

- Configure OpenGL Context
- Initialize Scene:
  - Init Vertex Data
  - Init Texture Data
  - Init Shaders
  - Init Shader Program
  - Init Camera Matrix
- Enter Render Loop:
  - Update Scene
  - Draw Scene
  - Repeat



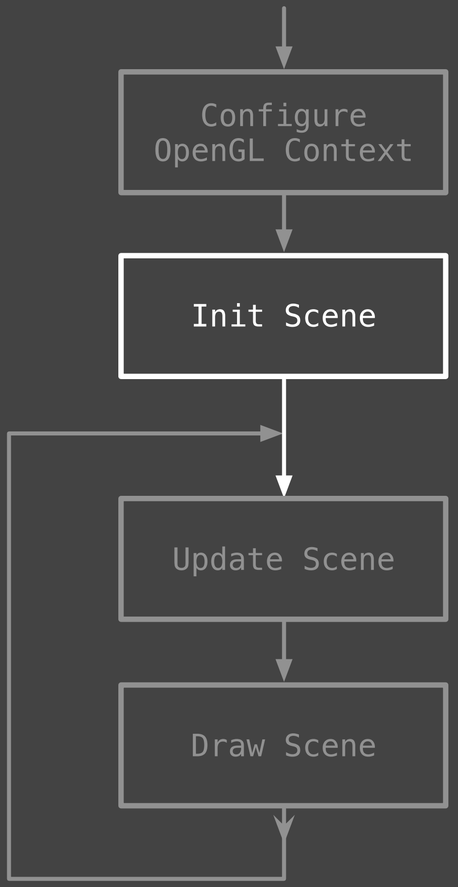
Configure Context

```
func ConfigureContext() (int32,int32) {

    Notice("create context..")
    err := piglet.CreateContext()
    if err != nil {
        Error("fail create context: %s",err)
    }
    width,height := piglet.GetDisplaySize()
    Notice("display: %dx%d",width,height)

    piglet.MakeCurrent()
    gl.InitWithProcAddrFunc( piglet.GetProcAddress )

    return width,height
}
```



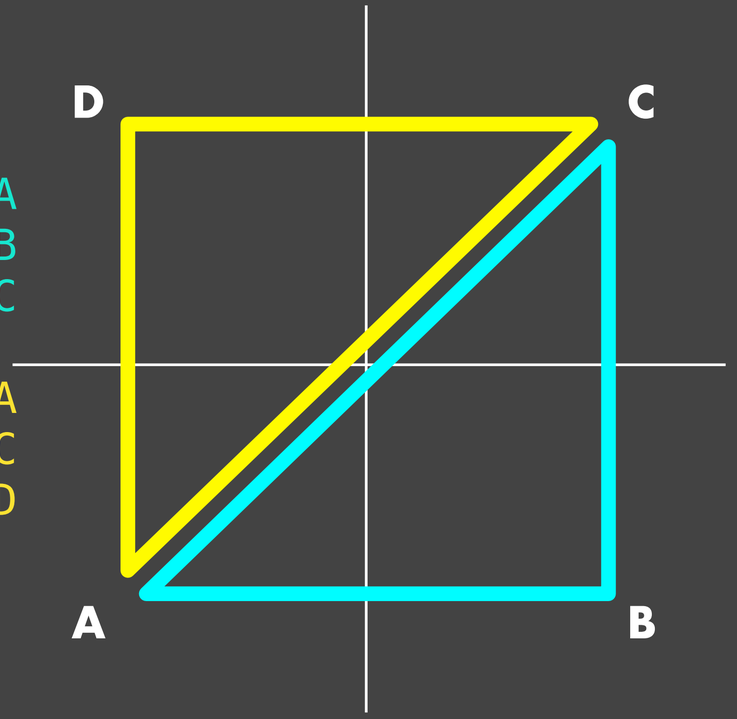
Initialize Scene

```
func InitScene(width,height int32) (uint32,uint32,uint32,mgl32.Mat4) {
    Notice("init scene..")

    cameraMatrix := InitCameraMatrix(float32(width),float32(height))
    vertexBuffer := InitVertexBuffer()
    textureName := InitTextureName()

    vertexShader := InitShader(VertexSource,gl.VERTEX_SHADER)
    fragmentShader := InitShader(FragmentSource, gl.FRAGMENT_SHADER)
    shaderProgram := InitShaderProgram(vertexShader, fragmentShader)

    gl.Viewport(0, 0, width, height)
    gl.ClearColor(0.25, 0.25, 0.25, 1.0)

    startTime := time.Now()

    return shaderProgram,vertexBuffer,textureName,cameraMatrix,startTime
}
```



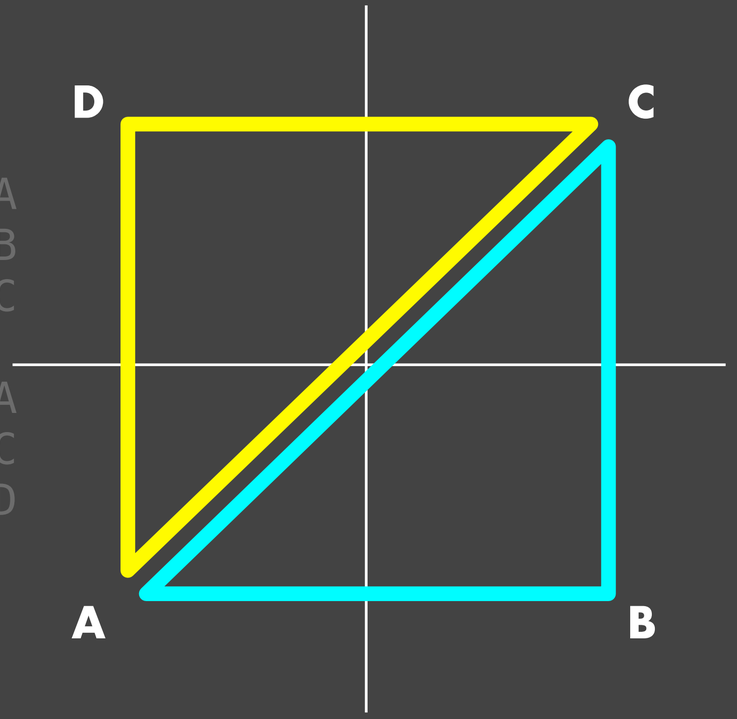
Init Vertex Data

```
var VertexData = []float32{
//    position              texCoord          color
//    x,    y,    z,        s,    t,          r,   g,   b,   a,
   -1.0, -1.0,   0.,       0.,   1.,         0.,  0.,  0.,  1.,    // A
    1.0, -1.0,   0.,       1.,   1.,         0.,  1.,  1.,  1.,    // B
    1.0,  1.0,   0.,       1.,   0.,         1.,  1.,  0.,  1.,    // C

   -1.0, -1.0,   0.,       0.,   1.,         0.,  0.,  0.,  1.,    // A
    1.0,  1.0,   0.,       1.,   0.,         1.,  1.,  0.,  1.,    // C
   -1.0,  1.0,   0.,       0.,   0.,         1.,  0.,  1.,  1.,    // D
}

func InitVertexBuffer() uint32 {
    var vertexBuffer uint32
    gl.GenBuffers(1,&vertexBuffer)
    gl.BindBuffer(gl.ARRAY_BUFFER, vertexBuffer)
    gl.BufferData(gl.ARRAY_BUFFER, len(VertexData)*4, gl.Ptr(VertexData), gl.STATIC_DRAW)

    if gl.GetError() != gl.NO_ERROR {
        Error("fail init buffer")
    }
    return vertexBuffer
}
```



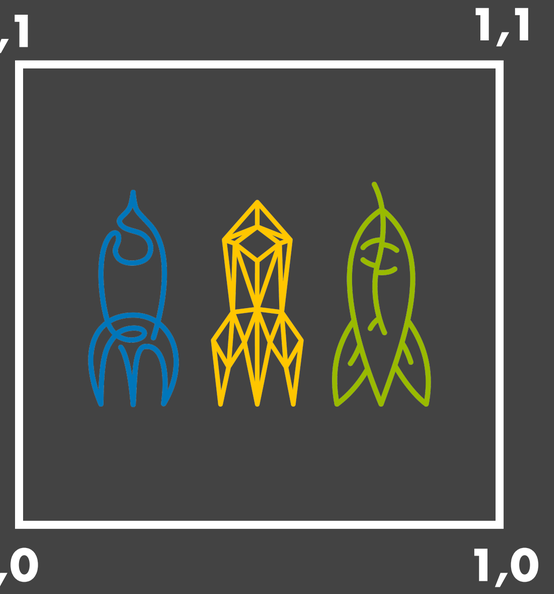
Init Vertex Data

```
var VertexData = []float32{
//    position              texCoord          color
//     x,    y,    z,        s,    t,          r,   g,   b,   a,
   -1.0, -1.0,   0.,       0.,   1.,         0.,  0.,  0.,  1.,    // A
    1.0, -1.0,   0.,       1.,   1.,         0.,  1.,  1.,  1.,    // B
    1.0,  1.0,   0.,       1.,   0.,         1.,  1.,  0.,  1.,    // C

   -1.0, -1.0,   0.,       0.,   1.,         0.,  0.,  0.,  1.,    // A
    1.0,  1.0,   0.,       1.,   0.,         1.,  1.,  0.,  1.,    // C
   -1.0,  1.0,   0.,       0.,   0.,         1.,  0.,  1.,  1.,    // D
}

func InitVertexBuffer() uint32 {
    var vertexBuffer uint32
    gl.GenBuffers(1,&vertexBuffer)
    gl.BindBuffer(gl.ARRAY_BUFFER, vertexBuffer)
    gl.BufferData(gl.ARRAY_BUFFER, len(VertexData)*4, gl.Ptr(VertexData), gl.STATIC_DRAW)

    if gl.GetError() != gl.NO_ERROR {
        Error("fail init buffer")
    }
    return vertexBuffer
}
```



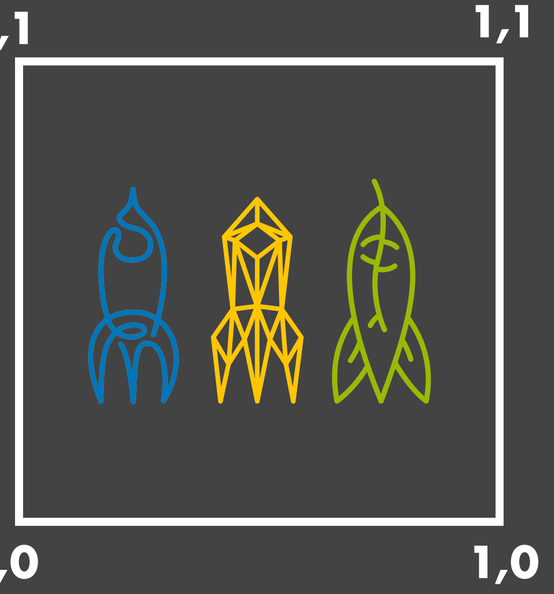
Init Texture Data

```
func InitTextureName() uint32 {
    var textureName uint32

    reader := base64.NewDecoder(base64.StdEncoding, strings.NewReader( TextureData ) )
    img,_,_ := image.Decode(reader)
    textureRGBA := image.NewRGBA( img.Bounds() )
    draw.Draw(textureRGBA, textureRGBA.Bounds(), img, image.Point{0,0}, draw.Src)
    w,h := int32(textureRGBA.Rect.Size().X), int32(textureRGBA.Rect.Size().Y)
    ptr := gl.Ptr(textureRGBA.Pix)

    gl.ActiveTexture(gl.TEXTURE0)
    gl.GenTextures(1, &textureName)
    gl.BindTexture(gl.TEXTURE_2D, textureName)
    gl.TexParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR)
    gl.TexParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR)
    gl.TexImage2D(gl.TEXTURE_2D,0,gl.RGBA,w,h,0,gl.RGBA,gl.UNSIGNED_BYTE,ptr)

    if gl.GetError() != gl.NO_ERROR {
        Error("fail init texture")
    }
    return textureName
}
```



Init Texture Data

```
func InitTextureName() uint32 {
    var textureName uint32

    reader := base64.NewDecoder(base64.StdEncoding, strings.NewReader( TextureData ) )
    img,_,_ := image.Decode(reader)
    textureRGBA := image.NewRGBA( img.Bounds() )
    draw.Draw(textureRGBA, textureRGBA.Bounds(), img, image.Point{0,0}, draw.Src)
    w,h := int32(textureRGBA.Rect.Size().X), int32(textureRGBA.Rect.Size().Y)
    ptr := gl.Ptr(textureRGBA.Pix)

    gl.ActiveTexture(gl.TEXTURE0)
    gl.GenTextures(1, &textureName)
    gl.BindTexture(gl.TEXTURE_2D, textureName)
    gl.TexParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR)
    gl.TexParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR)
    gl.TexImage2D(gl.TEXTURE_2D,0,gl.RGBA,w,h,0,gl.RGBA,gl.UNSIGNED_BYTE,ptr)

    if gl.GetError() != gl.NO_ERROR {
        Error("fail init texture")
    }
    return textureName
}
```



Init Vertex Shader

```
const VertexSource = `
uniform mat4 camera;
attribute vec3 position;
attribute vec2 texCoord; varying vec2 vTexCoord;
attribute vec4 color;    varying vec4 vColor;

void main() {
    vTexCoord = texCoord;
    vColor = color;
    gl_Position = camera * vec4(position, 1);
}`
func InitShader(source string, xtype uint32) uint32 {
    var status int32
    shader := gl.CreateShader( xtype )
    src, free := gl.Strs(source+"\x00")
    defer free()
    gl.ShaderSource( shader, 1, src, nil )
    gl.CompileShader( shader )
    gl.GetShaderiv(shader,gl.COMPILE_STATUS,&status)
    if status == gl.FALSE {
        Error("fail compile shader")
    }
    return shader
}
```

- pass on texCoord and color vertex attributes as varying
- final position is position vertex attribute multiplied by camera matrix

Init Vertex Shader

```
const VertexSource = `
uniform mat4 camera;
attribute vec3 position;
attribute vec2 texCoord; varying vec2 vTexCoord;
attribute vec4 color;    varying vec4 vColor;

void main() {
    vTexCoord = texCoord;
    vColor = color;
    gl_Position = camera * vec4(position, 1);
}`
func InitShader(source string, xtype uint32) uint32 {
    var status int32
    shader := gl.CreateShader( xtype )
    src, free := gl.Strs(source+"\x00")
    defer free()
    gl.ShaderSource( shader, 1, src, nil )
    gl.CompileShader( shader )
    gl.GetShaderiv(shader,gl.COMPILE_STATUS,&status)
    if status == gl.FALSE {
        Error("fail compile shader")
    }
    return shader
}
```

Init Fragment Shader

```
const FragmentSource = `
uniform sampler2D texture;
varying vec2 vTexCoord;
varying vec4 vColor;

void main() {
    vec4 texColor = texture2D(texture,vTexCoord);
    gl_FragColor = vec4( vColor.rgb, 1. - texColor.a );
}`

func InitShader(source string, xtype uint32) uint32 {
    var status int32
    shader := gl.CreateShader( xtype )
    src, free := gl.Strs(source+"\x00")
    defer free()
    gl.ShaderSource( shader, 1, src, nil )
    gl.CompileShader( shader )
    gl.GetShaderiv(shader,gl.COMPILE_STATUS,&status)
    if status == gl.FALSE {
        Error("fail compile shader")
    }
    return shader
}
```

- sample texture at texture coordinate to get color from texture
- final color is RGB from color varying and transparency from texture

Init Fragment Shader

```
const FragmentSource = `
uniform sampler2D texture;
varying vec2 vTexCoord;
varying vec4 vColor;

void main() {
    vec4 texColor = texture2D(texture,vTexCoord);
    gl_FragColor = vec4( vColor.rgb, 1. - texColor.a );
}`

func InitShader(source string, xtype uint32) uint32 {
    var status int32
    shader := gl.CreateShader( xtype )
    src, free := gl.Strs(source+"\x00")
    defer free()
    gl.ShaderSource( shader, 1, src, nil )
    gl.CompileShader( shader )
    gl.GetShaderiv(shader,gl.COMPILE_STATUS,&status)
    if status == gl.FALSE {
        Error("fail compile shader")
    }
    return shader
}
```

Init Shader Program

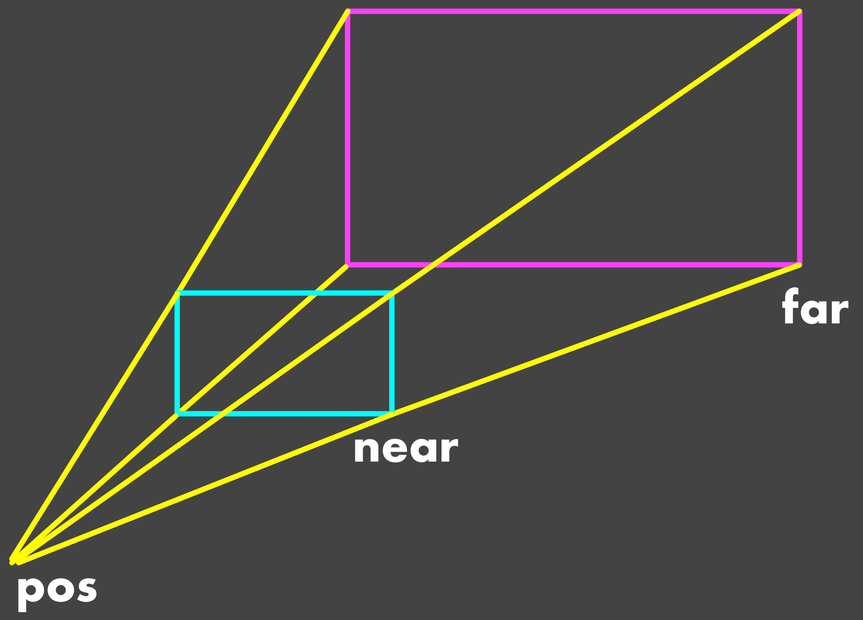
```
func InitShaderProgram(vertexShader,fragmentShader uint32) uint32 {
    var status int32
    shaderProgram := gl.CreateProgram()
    gl.AttachShader( shaderProgram, vertexShader)
    gl.AttachShader( shaderProgram, fragmentShader)
    gl.LinkProgram( shaderProgram )
    gl.GetProgramiv(shaderProgram, gl.LINK_STATUS, &status);
    if status == gl.FALSE {
        Error("fail link shader program")
    }

    ptr := gl.GetAttribLocation(shaderProgram, gl.Str("position\x00") )
    gl.EnableVertexAttribArray( uint32(ptr) )
    gl.VertexAttribPointer( uint32(ptr), 3, gl.FLOAT, false, (3+2+4)*4, gl.PtrOffset(0*4) )
    ptr = gl.GetAttribLocation(shaderProgram, gl.Str("texCoord\x00") )
    gl.EnableVertexAttribArray( uint32(ptr) )
    gl.VertexAttribPointer( uint32(ptr), 2, gl.FLOAT, false, (3+2+4)*4, gl.PtrOffset(3*4) )
    ptr = gl.GetAttribLocation(shaderProgram, gl.Str("color\x00") )
    gl.EnableVertexAttribArray( uint32(ptr) )
    gl.VertexAttribPointer( uint32(ptr), 4, gl.FLOAT, false, (3+2+4)*4, gl.PtrOffset((3+2)*4) )

    return shaderProgram
}
```

Init Shader Program

```
func InitShaderProgram(vertexShader,fragmentShader uint32) uint32 {
    var status int32
    shaderProgram := gl.CreateProgram()
    gl.AttachShader( shaderProgram, vertexShader)
    gl.AttachShader( shaderProgram, fragmentShader)
    gl.LinkProgram( shaderProgram )
    gl.GetProgramiv(shaderProgram, gl.LINK_STATUS, &status);
    if status == gl.FALSE {
        Error("fail link shader program")
    }

    ptr := gl.GetAttribLocation(shaderProgram, gl.Str("position\x00") )
    gl.EnableVertexAttribArray( uint32(ptr) )
    gl.VertexAttribPointer( uint32(ptr), 3, gl.FLOAT, false, (3+2+4)*4, gl.PtrOffset(0*4) )
    ptr = gl.GetAttribLocation(shaderProgram, gl.Str("texCoord\x00") )
    gl.EnableVertexAttribArray( uint32(ptr) )
    gl.VertexAttribPointer( uint32(ptr), 2, gl.FLOAT, false, (3+2+4)*4, gl.PtrOffset(3*4) )
    ptr = gl.GetAttribLocation(shaderProgram, gl.Str("color\x00") )
    gl.EnableVertexAttribArray( uint32(ptr) )
    gl.VertexAttribPointer( uint32(ptr), 4, gl.FLOAT, false, (3+2+4)*4, gl.PtrOffset((3+2)*4) )

    return shaderProgram
}
```

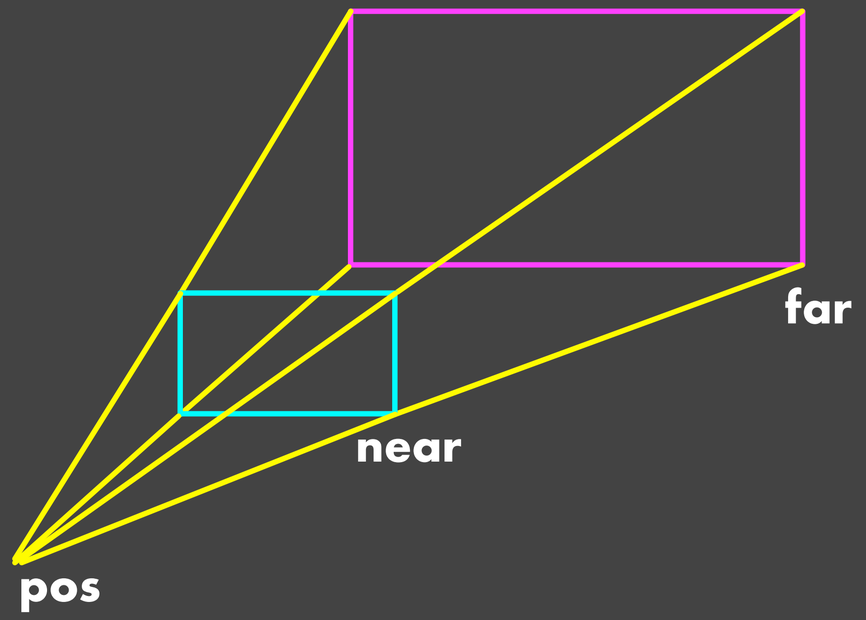
Init Camera Matrix

```
func InitCameraMatrix(width,height float32) mgl32.Mat4 {

    fov, ratio := mgl32.DegToRad(45.), width/height
    near,far := float32(0.01), float32(10.0)
    projection := mgl32.Perspective(fov, ratio, near, far)

    pos := mgl32.Vec3{0,0,5}
    lookat := mgl32.Vec3{0,0,0}
    up := mgl32.Vec3{0,1,0}
    view := mgl32.LookAtV( pos, lookat, up )

    var cameraMatrix = mgl32.Ident4()
    cameraMatrix = cameraMatrix.Mul4( projection )
    cameraMatrix = cameraMatrix.Mul4( view )
    return cameraMatrix
}
```



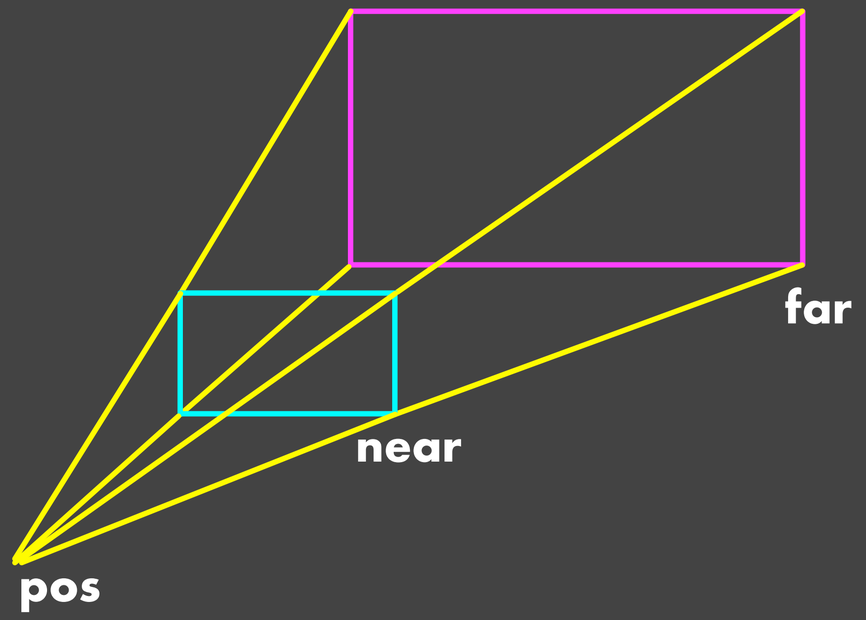
Init Camera Matrix

```
func InitCameraMatrix(width,height float32) mgl32.Mat4 {

    fov, ratio := mgl32.DegToRad(45.), width/height
    near,far := float32(0.01), float32(10.0)
    projection := mgl32.Perspective(fov, ratio, near, far)

    pos := mgl32.Vec3{0,0,5}
    lookat := mgl32.Vec3{0,0,0}
    up := mgl32.Vec3{0,1,0}
    view := mgl32.LookAtV( pos, lookat, up )

    var cameraMatrix = mgl32.Ident4()
    cameraMatrix = cameraMatrix.Mul4( projection )
    cameraMatrix = cameraMatrix.Mul4( view )
    return cameraMatrix
}
```

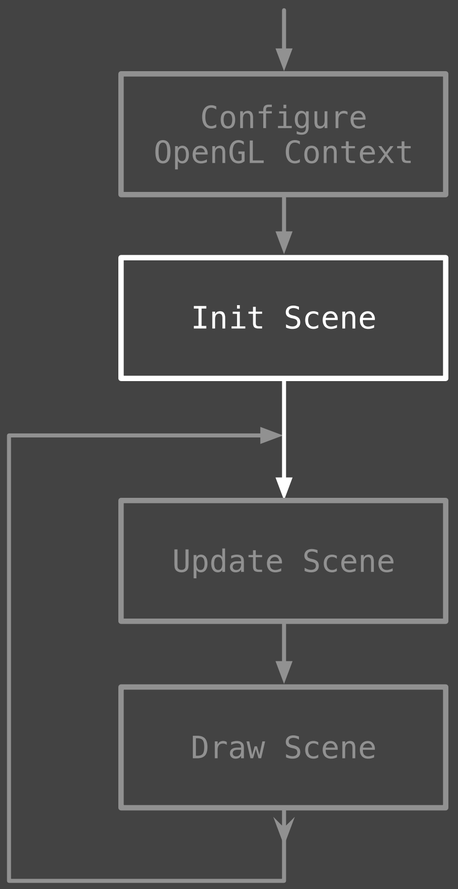


Init Camera Matrix

```
func InitCameraMatrix(width,height float32) mgl32.Mat4 {

    fov, ratio := mgl32.DegToRad(45.), width/height
    near,far := float32(0.01), float32(10.0)
    projection := mgl32.Perspective(fov, ratio, near, far)

    pos := mgl32.Vec3{0,0,5}
    lookat := mgl32.Vec3{0,0,0}
    up := mgl32.Vec3{0,1,0}
    view := mgl32.LookAtV( pos, lookat, up )

    var cameraMatrix = mgl32.Ident4()
    cameraMatrix = cameraMatrix.Mul4( projection )
    cameraMatrix = cameraMatrix.Mul4( view )
    return cameraMatrix
}
```

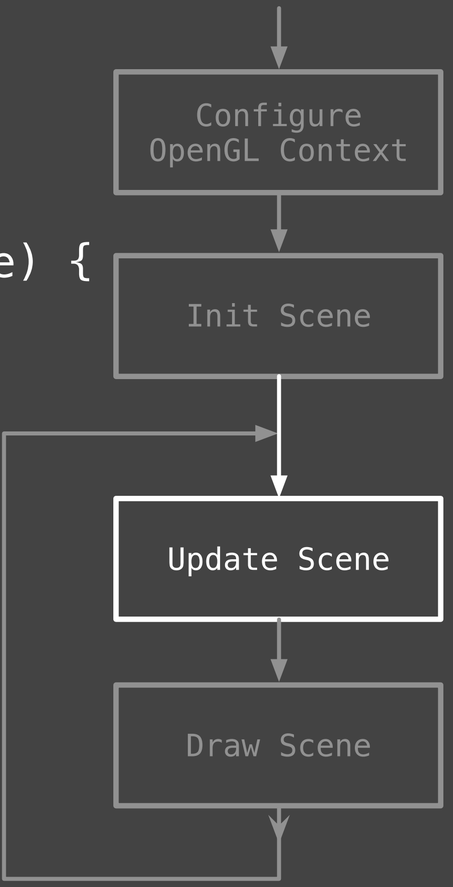


Scene Initialized



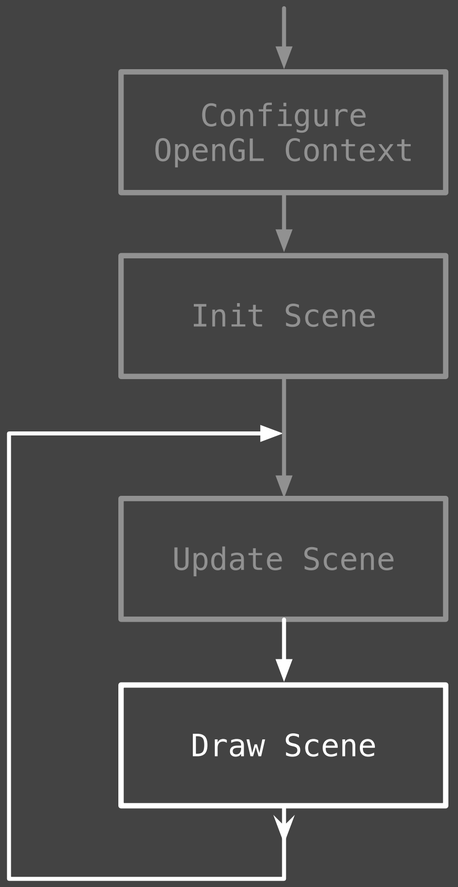
Update Scene

```
func UpdateScene(shaderProgram uint32, camera mgl32.Mat4, startTime time.Time) {

    clock := float32( time.Now().Sub( startTime ).Seconds() )

    angle := PI/3. * Sin( clock )
    cam := camera.Mul4( mgl32.HomogRotate3DY( angle ) )

    gl.UseProgram( shaderProgram )
    ptr := gl.GetUniformLocation(shaderProgram, gl.Str("camera\x00") )
    gl.UniformMatrix4fv(ptr, 1, false, &cam[0])

}
```

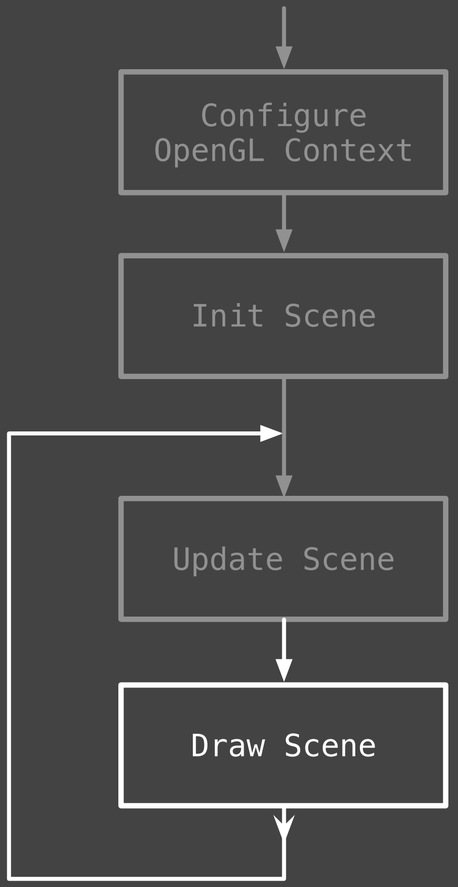


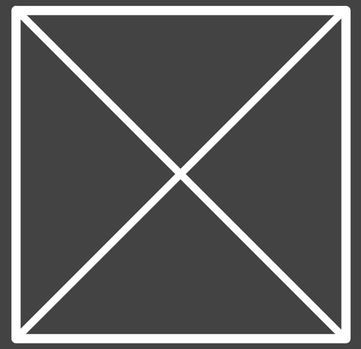
Draw Scene

```
func DrawScene(shaderProgram,vertexBuffer,textureName uint32) {

    gl.Clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT )

    gl.UseProgram( shaderProgram )
    gl.BindBuffer(gl.ARRAY_BUFFER, vertexBuffer)
    gl.BindTexture(gl.TEXTURE_2D, textureName)

    gl.DrawArrays(gl.TRIANGLES, 0, 2*3)

    err := gl.GetError()
    if err != gl.NO_ERROR {
        Error("fail draw: %s", gl.ErrorString(err))
    }

}
```



Scene drawn
(now do it again)





Video not Found :/

glEnd();

Resources

- www.khronos.org/registry/OpenGL-Refpages/es2.0/
- www.khronos.org/files/opengles20-reference-card.pdf
- OpenGL ES 2.0 Programming Guide
  by Munshi, Ginsburg, Shreiner
- 'The Red Book' - Classic, but mostly outdated now