

Special Effects

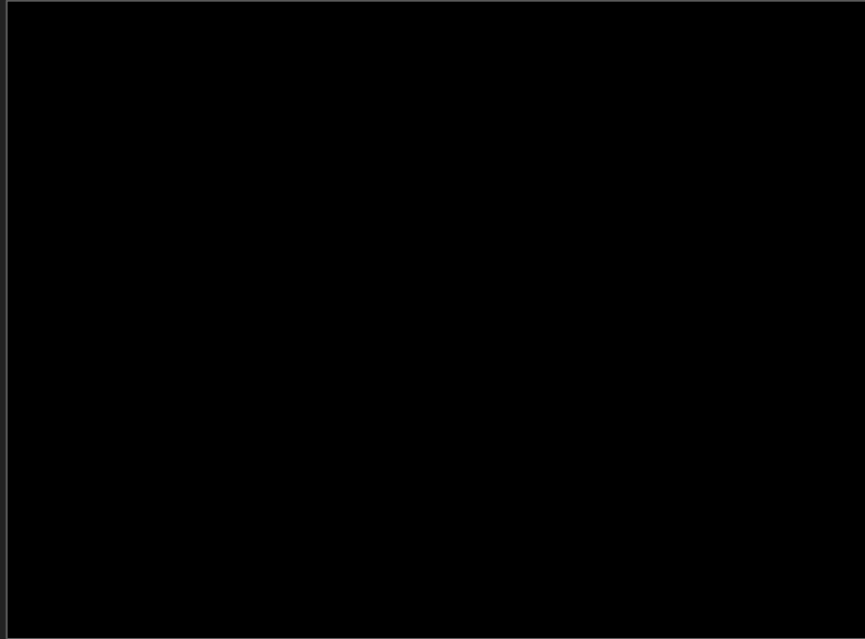
We can add some polish to our
games with some simple
special effects.

We can code these simple effects by overlaying a large square of a solid color then adjusting the alpha or the position.

Fade In



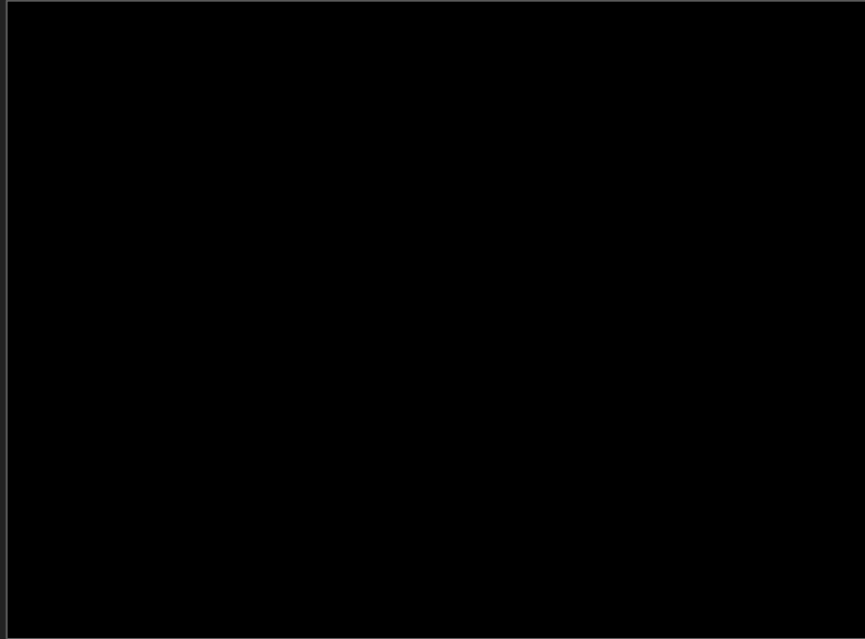
Fade Out



Slide Up



Slide Down



Grow



Shrink



Let's add a new class for
handling effects.

It will have a Constructor,
Start, Update and Render.

New Effects Class

Effects.h

Effects.cpp

Effects.h

```
#pragma once
#define GL_SILENCE_DEPRECATION
#ifdef _WINDOWS
#include <GL/glew.h>
#endif

#define GL_GLEXT_PROTOTYPES 1
#include <vector>
#include <math.h>
#include <SDL.h>
#include <SDL_opengl.h>
#include "glm/mat4x4.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "ShaderProgram.h"

enum EffectType { NONE, FADEIN };

class Effects {
    ShaderProgram program;
    float alpha;
    EffectType currentEffect;

public:
    Effects(glm::mat4 projectionMatrix, glm::mat4 viewMatrix);
    void DrawOverlay();
    void Start(EffectType effectType);
    void Update(float deltaTime);
    void Render();
};
```

Effects.cpp

```
#include "Effects.h"

Effects::Effects(glm::mat4 projectionMatrix, glm::mat4 viewMatrix)
{
    // Non textured Shader
    program.Load("shaders/vertex.glsl", "shaders/fragment.glsl");
    program.SetProjectionMatrix(projectionMatrix);
    program.SetViewMatrix(viewMatrix);

    currentEffect = NONE;
    alpha = 0;
}

void Effects::DrawOverlay()
{
    glUseProgram(program.programID);
    float vertices[] = { -0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5 };
    glVertexAttribPointer(program.positionAttribute, 2, GL_FLOAT, false, 0, vertices);
    glEnableVertexAttribArray(program.positionAttribute);
    glDrawArrays(GL_TRIANGLES, 0, 6);
    glDisableVertexAttribArray(program.positionAttribute);
}
```

Effects.cpp

```
void Effects::Start(EffectType effectType)
{
    currentEffect = effectType;

    switch (currentEffect) {
        case NONE:
            break;

        case FADEIN:
            break;
    }
}

void Effects::Update(float deltaTime)
{
    switch (currentEffect) {
        case NONE:
            break;

        case FADEIN:
            break;
    }
}

void Effects::Render()
{
    switch (currentEffect) {
        case NONE:
            return;
            break;

        case FADEIN:
            break;
    }
}
```

Update main.cpp

```
// Add the include
#include "Effects.h"

// Add an Effects variable
Effects *effects;

// Initialize effects inside Initialize
effects = new Effects(projectionMatrix, viewMatrix);

// Start FADEIN effect
effects->Start(FADEIN);

// Add to Update
effects->Update(FIXED_TIMESTEP);

// Add to top of Render
glUseProgram(program.programID);

// Add to end of Render
effects->Render();
```

Checkpoint:

We should be able to compile
and everything should
look the same (for now).

Let's create the
Fade In effect together!

Let's add Fade Out

(You can re-use a lot of code from Fade In)

How do we make these effects
run slower or faster?

(let's code it)

Let's add more effects!
Grow and Shrink.

One more effect:
“Camera Shake”

We can randomly translate the viewMatrix

```
// We'll need this code  
float max = 0.1f;  
float min = -0.1f;  
float r = ((float)rand() / RAND_MAX) * (max - min) + min;  
viewOffset = glm::vec3(r, r, 0);
```

Shaders!

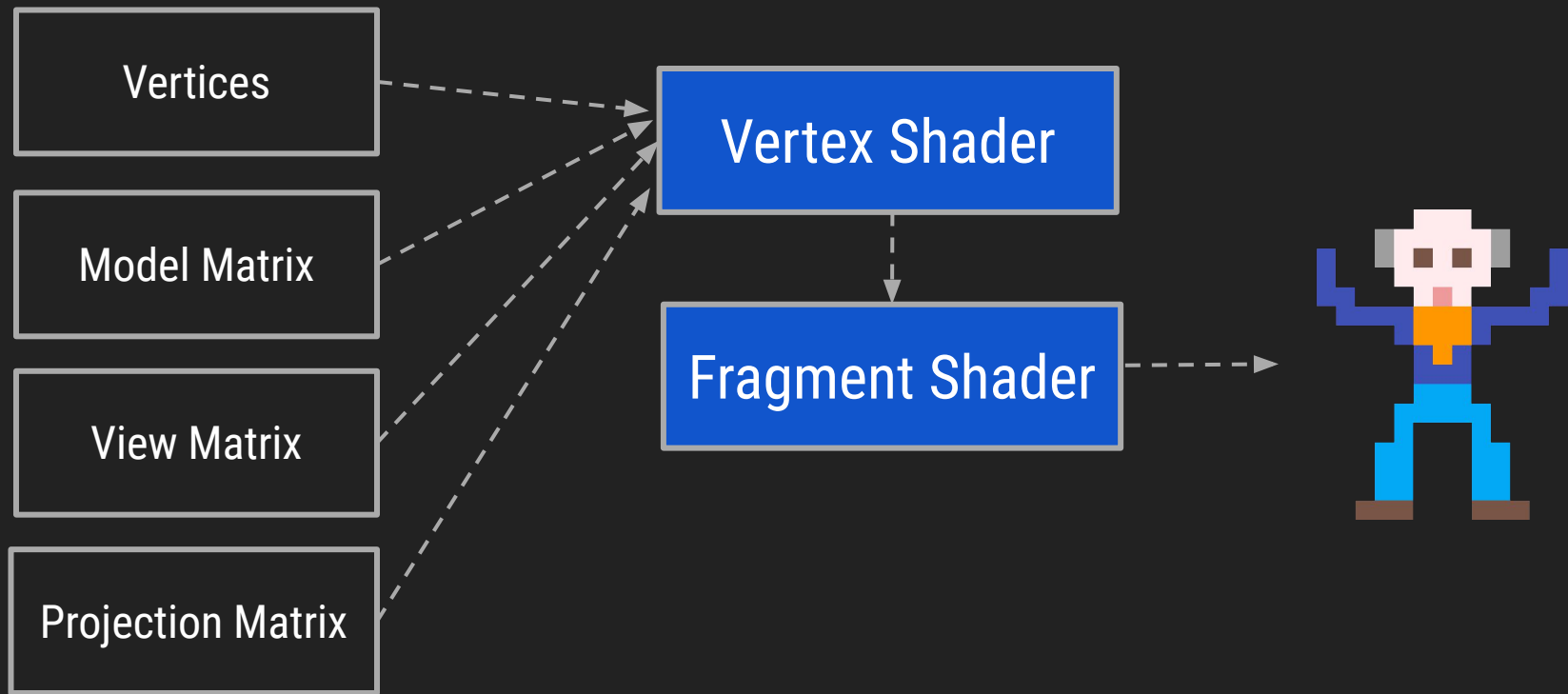
What are these things?

What can we do with them?

We have lots of code that uses them:

```
program.Load("shaders/vertex_textured.glsl",  
            "shaders/fragment_textured.glsl");  
  
program.SetProjectionMatrix(projectionMatrix);  
  
program.SetViewMatrix(viewMatrix);  
  
program.SetColor(1.0f, 1.0f, 1.0f, 1.0f);  
  
program.SetModelMatrix(modelMatrix);  
  
glUseProgram(program.programID);
```


The GPU Pipeline



The (complete) Shader program
is made from a
Vertex and a Fragment Shader

```
program.Load(  
    "shaders/vertex_textured.glsl",  
    "shaders/fragment_textured.glsl");
```

Vertex Shader

Translates vertices to screen positions.

Look at the code in our games:

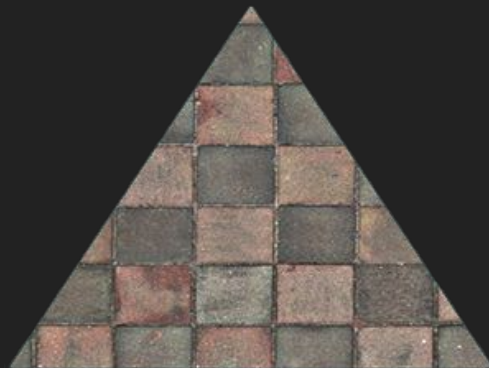
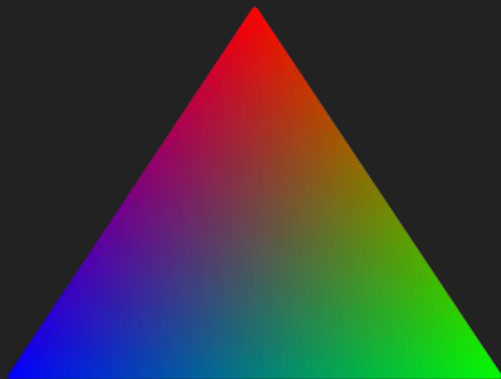
```
program.SetProjectionMatrix(projectionMatrix);  
program.SetViewMatrix(viewMatrix);  
program.SetModelMatrix(modelMatrix);
```

Look at shaders/vertex_textured.glsl

```
void main()  
{  
    vec4 p = viewMatrix * modelMatrix * position;  
    texCoordVar = texCoord;  
    gl_Position = projectionMatrix * p;  
}
```

Fragment Shader

For each pixel, determines what color to draw.
Interpolated by distance from vertices.
(might also grab a pixel/color from a texture)



Look at shaders/fragment.glsl

```
uniform vec4 color;
```

```
void main() {  
    gl_FragColor = color;  
}
```

Look at shaders/fragment_textured.glsl

```
uniform sampler2D diffuse;  
varying vec2 texCoordVar;
```

```
void main() {  
    gl_FragColor = texture2D(diffuse, texCoordVar);  
}
```

These Shaders are made
in a C like language
called GLSL

Let's take a closer look
at `vertex_textured.glsl`


vertex_textured.glsl

```
attribute vec4 position;  
attribute vec2 texCoord;  
  
uniform mat4 modelMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 projectionMatrix;  
  
varying vec2 texCoordVar;  
  
void main()  
{  
    vec4 p = viewMatrix * modelMatrix * position;  
    texCoordVar = texCoord;  
    gl_Position = projectionMatrix * p;  
}
```

vertex_textured.glsl

```
attribute vec4 position;  
attribute vec2 texCoord;
```

Vertices and Texture
Coordinates



```
uniform mat4 modelMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 projectionMatrix;
```

```
varying vec2 texCoordVar;
```

```
void main()
```

```
{
```

```
    vec4 p = viewMatrix * modelMatrix * position;
```

```
    texCoordVar = texCoord;
```

```
    gl_Position = projectionMatrix * p;
```

```
}
```

vertex_textured.glsl

```
attribute vec4 position;  
attribute vec2 texCoord;
```

```
uniform mat4 modelMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 projectionMatrix;
```

Variables we can set
directly from our code.

```
varying vec2 texCoordVar;
```

```
void main()  
{  
    vec4 p = viewMatrix * modelMatrix * position;  
    texCoordVar = texCoord;  
    gl_Position = projectionMatrix * p;  
}
```

vertex_textured.glsl

```
attribute vec4 position;  
attribute vec2 texCoord;
```

```
uniform mat4 modelMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 projectionMatrix;
```

```
varying vec2 texCoordVar;
```

A variable that is passed
from the Vertex shader to
the Fragment shader.

```
void main()
```

```
{
```

```
    vec4 p = viewMatrix * modelMatrix * position;
```

```
    texCoordVar = texCoord;
```

```
    gl_Position = projectionMatrix * p;
```

```
}
```

vertex_textured.glsl

```
attribute vec4 position;  
attribute vec2 texCoord;  
  
uniform mat4 modelMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 projectionMatrix;  
  
varying vec2 texCoordVar;  
  
void main()  
{  
    vec4 p = viewMatrix * modelMatrix * position;  
    texCoordVar = texCoord;  
    gl_Position = projectionMatrix * p;  
}
```

The position of the vertex is calculated.
Texture coordinate is just passed along.
`gl_position` must be set.

Let's take a closer look at
fragment_textured.glsl

fragment_textured.glsl

```
uniform sampler2D diffuse;  
  
varying vec2 texCoordVar;  
  
void main() {  
    gl_FragColor = texture2D(diffuse, texCoordVar);  
}
```

fragment_textured.glsl

```
uniform sampler2D diffuse;
```

```
varying vec2 texCoordVar;
```

```
void main() {
```

```
    gl_FragColor = texture2D(diffuse, texCoordVar);
```

```
}
```



A sampler is a texture.


fragment_textured.glsl

```
uniform sampler2D diffuse;  
  
varying vec2 texCoordVar;  
  
void main() {  
    gl_FragColor = texture2D(diffuse, texCoordVar);  
}
```

← -----
The coordinate passed in
from the Vertex shader.

fragment_textured.glsl

```
uniform sampler2D diffuse;  
  
varying vec2 texCoordVar;  
  
void main() {  
    gl_FragColor = texture2D(diffuse, texCoordVar);  
}
```



The color of the pixel is the color
from the texture based on the
texture coordinates

Let's Experiment!

Make a new file called `effects_textured.glsl`
copy the code from `fragment_textured.glsl`

Update `main.cpp`

```
program.Load("shaders/vertex_textured.glsl",  
            "shaders/effects_textured.glsl");
```

effects_textured.glsl

```
uniform sampler2D diffuse;

varying vec2 texCoordVar;

void main() {
    vec4 color = texture2D(diffuse, texCoordVar);
    gl_FragColor = vec4(color.r, 0, 0, color.a);
}
```

Let's run our program and see
what happens.

Then let's experiment with the
other colors.

We can create a
color inverter shader.

effects_textured.glsl

```
uniform sampler2D diffuse;

varying vec2 texCoordVar;

void main() {
    vec4 color = texture2D(diffuse, texCoordVar);
    gl_FragColor = vec4(1.0 - color.r, 1.0 - color.g, 1.0 - color.b, color.a);
}
```

How would we make a
grayscale shader?

effects_textured.glsl

This example averages the R, G, B values.

```
uniform sampler2D diffuse;

varying vec2 texCoordVar;

void main() {
    vec4 color = texture2D(diffuse, texCoordVar);
    float c = (color.r + color.g + color.b) / 3.0;
    gl_FragColor = vec4(c, c, c, color.a);
}
```

effects_textured.glsl

This example uses luminosity.

```
uniform sampler2D diffuse;

varying vec2 texCoordVar;

void main() {
    vec4 color = texture2D(diffuse, texCoordVar);
    vec3 luminance = vec3(dot(vec3(0.2126, 0.7152, 0.0722), color.rgb));
    gl_FragColor = vec4(luminance, color.a);
}
```

We can work with
luminance to create a
saturate / desaturate shader.

effects_textured.glsl

`mix(a, b, w)` performs a linear interpolation: $a*(1-w)+b*w$

```
uniform sampler2D diffuse;

varying vec2 texCoordVar;

void main() {
    vec4 color = texture2D(diffuse, texCoordVar);
    vec3 luminance = vec3(dot(vec3(0.2126, 0.7152, 0.0722), color.rgb));
    vec3 m = mix(luminance, color.rgb, 1.0);

    gl_FragColor = vec4(m, color.a);
}
```

Change the 1.0 to 2.0 to saturate, 0.5 to desaturate.

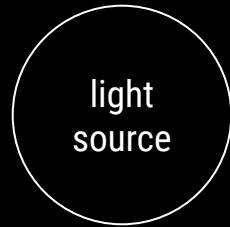
2D Lighting!



We can create a
shader to handle
a light in our scene.

For each pixel, we need to check the
distance from the light.
We decrease the brightness based on the
light's attenuation.

(It's related to the inverse square law.)



Lit Shader

Make new files called
`vertex_lit.glsl` and `fragment_lit.glsl`

Update `main.cpp`

```
program.Load("shaders/vertex_lit.glsl",  
            "shaders/fragment_lit.glsl");
```

vertex_lit.glsl

```
attribute vec4 position;  
attribute vec2 texCoord;  
  
uniform mat4 modelMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 projectionMatrix;  
  
varying vec2 texCoordVar;  
varying vec2 varPosition;  
  
void main()  
{  
    vec4 p = modelMatrix * position;  
    varPosition = vec2(p.x, p.y);  
    texCoordVar = texCoord;  
    gl_Position = projectionMatrix * viewMatrix * p;  
}
```

fragment_lit.glsl

```
uniform sampler2D diffuse;
uniform vec2 lightPosition;

varying vec2 texCoordVar;
varying vec2 varPosition;

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + (a * dist) + (b * dist * dist));
}

void main() {
    float brightness = attenuate(distance(lightPosition, varPosition), 1.0, 0.0);
    vec4 color = texture2D(diffuse, texCoordVar);
    gl_FragColor = vec4(color.rgb * brightness, color.a);
}
```

We need to add code so we can set the
position of the light in the shader.

ShaderProgram.h

```
// Add this method  
void SetLightPosition(glm::vec3 position);
```

```
// Add this variable  
GLuint lightPositionUniform;
```

ShaderProgram.cpp

```
// Add this inside of Load
lightPositionUniform = glGetUniformLocation(programID, "lightPosition");

// Add this method
void ShaderProgram::SetLightPosition(glm::vec3 position) {
    glUseProgram(programID);
    glUniform2f(lightPositionUniform, position.x, position.y);
}
```

main.cpp

```
// Add this inside of render  
program.SetLightPosition(currentScene->state.player.position);
```


Let's Code!

Add Lit Shaders

Update ShaderProgram.h

Update ShaderProgram.cpp

Update main.cpp