# 2D Graphics and Sprites

GAM200

DigiPen Institute of Technology - (c) 2021

# Dynamic Lights

# Dynamic Lights

- Lighting can be one of the main mechanics of your real time graphical project
  - Torch light
  - Shadow monsters
  - Dungeon with limited vision
  - & much more

# Dynamic Lights

- Basic lighting scene in 2D
- Affected by light’s color components and object’s color components (textures, colors, materials)
- Simplified
    - resultColor = objectColor * lightColor

# Dynamic Lights

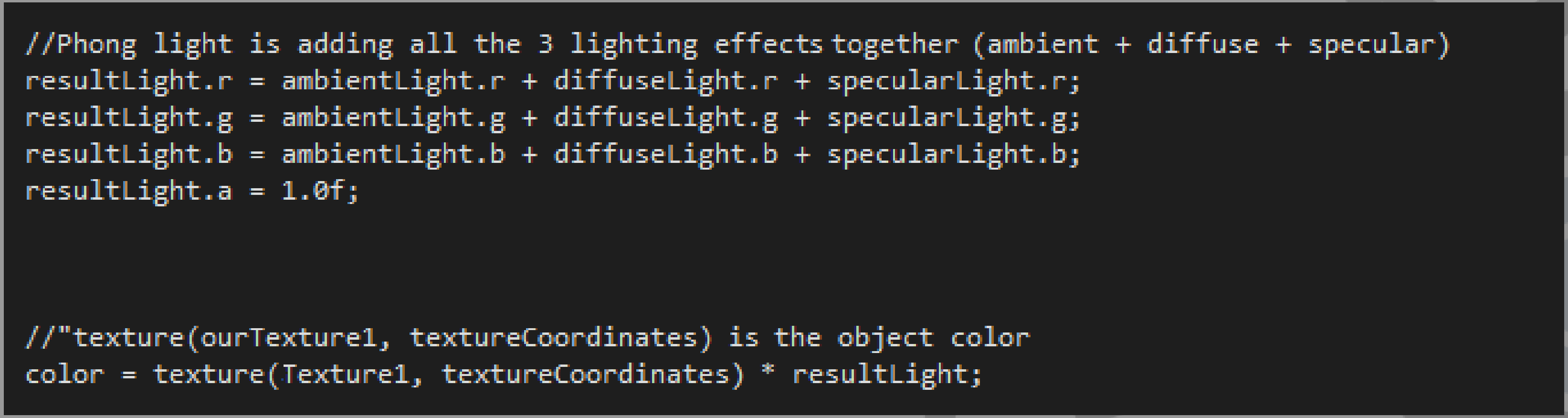
- A light can be a game object in your scene with special components
- LightComponent:
  - Color
  - Diffuse factor
  - Specular factor
  - Ambient factor
  - Attenuation
  - …

# Dynamic Lights

- Our lighting effects will be applied in the fragment/pixel shader
- We can write different fragment/pixel shaders to apply on different groups of objects

# Dynamic Lights

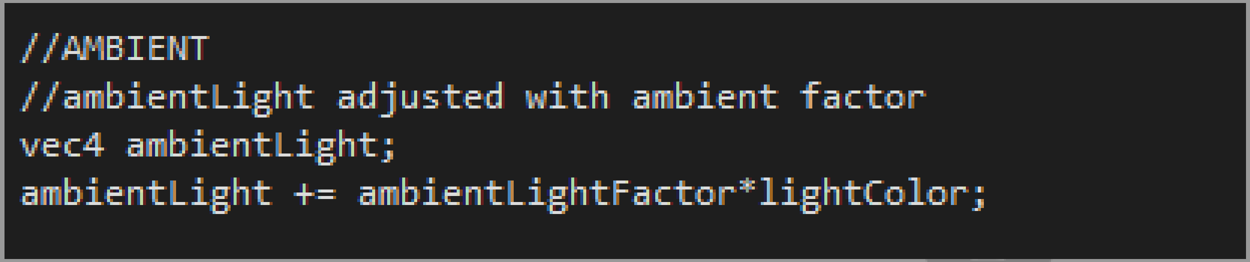
- As an example we will apply our basic *Phong* light effect



```
//Phong light is adding all the 3 lighting effects together (ambient + diffuse + specular)
resultLight.r = ambientLight.r + diffuseLight.r + specularLight.r;
resultLight.g = ambientLight.g + diffuseLight.g + specularLight.g;
resultLight.b = ambientLight.b + diffuseLight.b + specularLight.b;
resultLight.a = 1.0f;



//"texture(ourTexture1, textureCoordinates) is the object color
color = texture(Texture1, textureCoordinates) * resultLight;
```

# Dynamic Lights

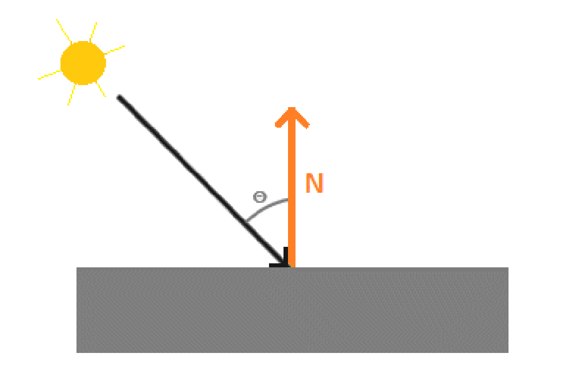
- Ambient Light
  - Is a scalar (percentage) of how much you want to apply of the original light’s color



```
//AMBIENT
//ambientLight adjusted with ambient factor
vec4 ambientLight;
ambientLight += ambientLightFactor*lightColor;
```

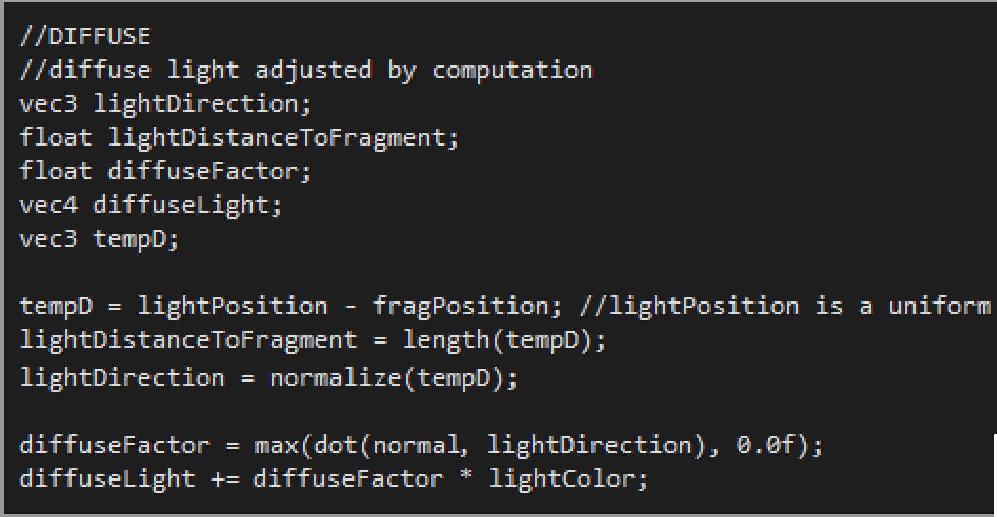
# Dynamic Lights

- Diffuse Light
  - Diffuse light computation involves the surface normal of where the light ray hits and the light direction gotten between the light position and the fragment lit
  - The intensity of the diffuse light is the dot product between the 2 vectors



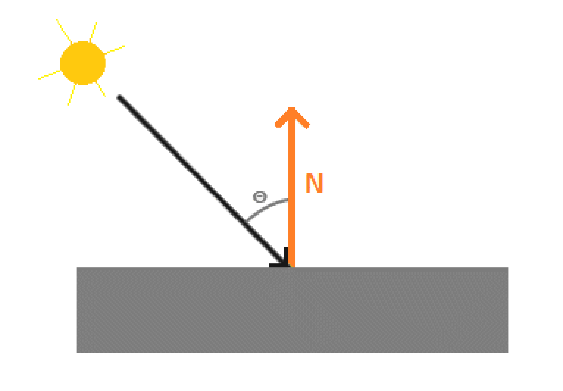
# Dynamic Lights

- Diffuse Light



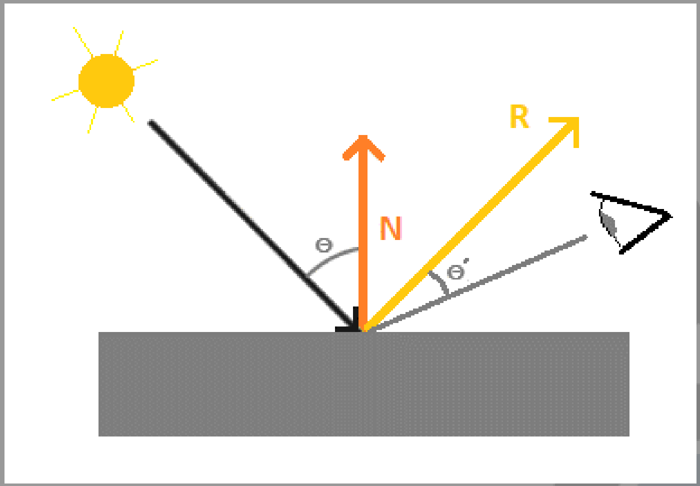
```
//DIFFUSE
//diffuse light adjusted by computation
vec3 lightDirection;
float lightDistanceToFragment;
float diffuseFactor;
vec4 diffuseLight;
vec3 tempD;

tempD = lightPosition - fragPosition; //lightPosition is a uniform variable read from c++
lightDistanceToFragment = length(tempD);
lightDirection = normalize(tempD);

diffuseFactor = max(dot(normal, lightDirection), 0.0f);
diffuseLight += diffuseFactor * lightColor;
```

- normal = vec3(0.0f,0.0f,1.0f) – hardcoded in 2D
- lightPosition has a greater z value from its 2D layer



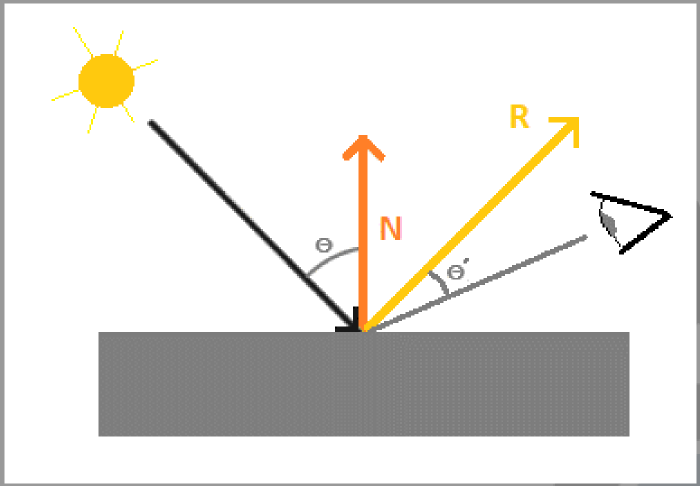
# Dynamic Lights

- Specular Light
  - Specular light is similar to diffuse light, with the difference of the eye (camera) position
  - In 2D this can be our viewport’s camera position



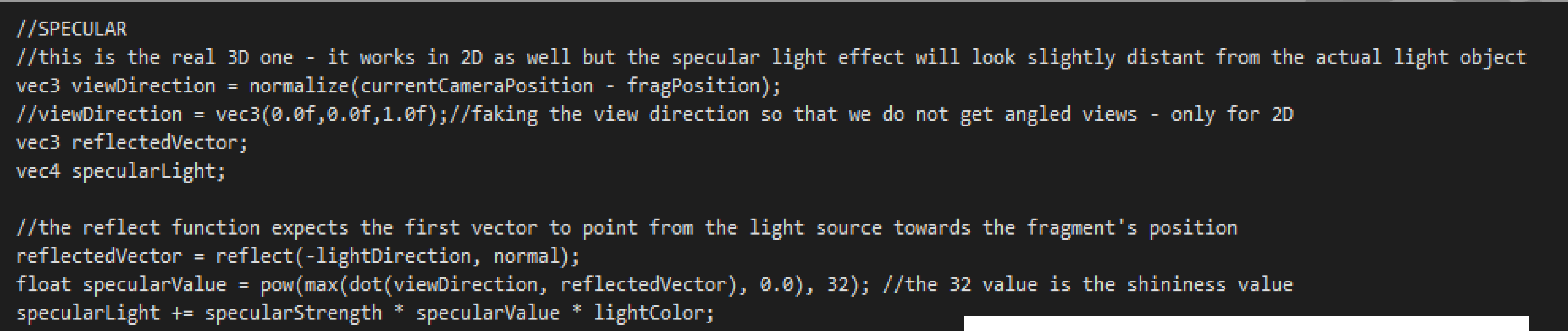
# Dynamic Lights

- Specular Light
  - Reflection vector needs to be calculated
  - Specular value needs to be calculated

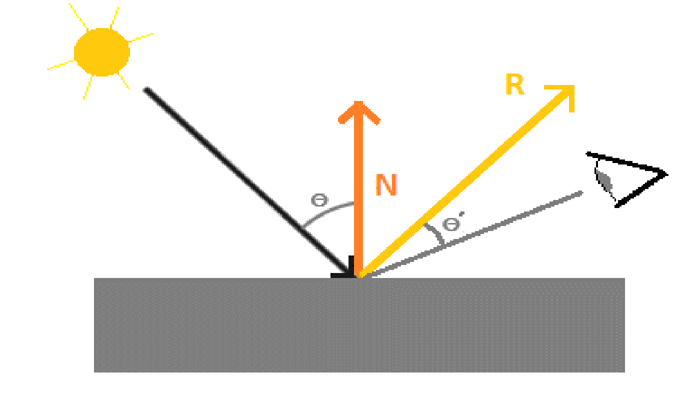


# Dynamic Lights

- Specular Light



```
//SPECULAR
//this is the real 3D one - it works in 2D as well but the specular light effect will look slightly distant from the actual light object
vec3 viewDirection = normalize(currentCameraPosition - fragPosition);
//viewDirection = vec3(0.0f,0.0f,1.0f);//faking the view direction so that we do not get angled views - only for 2D
vec3 reflectedVector;
vec4 specularLight;

//the reflect function expects the first vector to point from the light source towards the fragment's position
reflectedVector = reflect(-lightDirection, normal);
float specularValue = pow(max(dot(viewDirection, reflectedVector), 0.0), 32); //the 32 value is the shininess value
specularLight += specularStrength * specularValue * lightColor;
```



- specularStrength is a property of the light

# Dynamic Lights

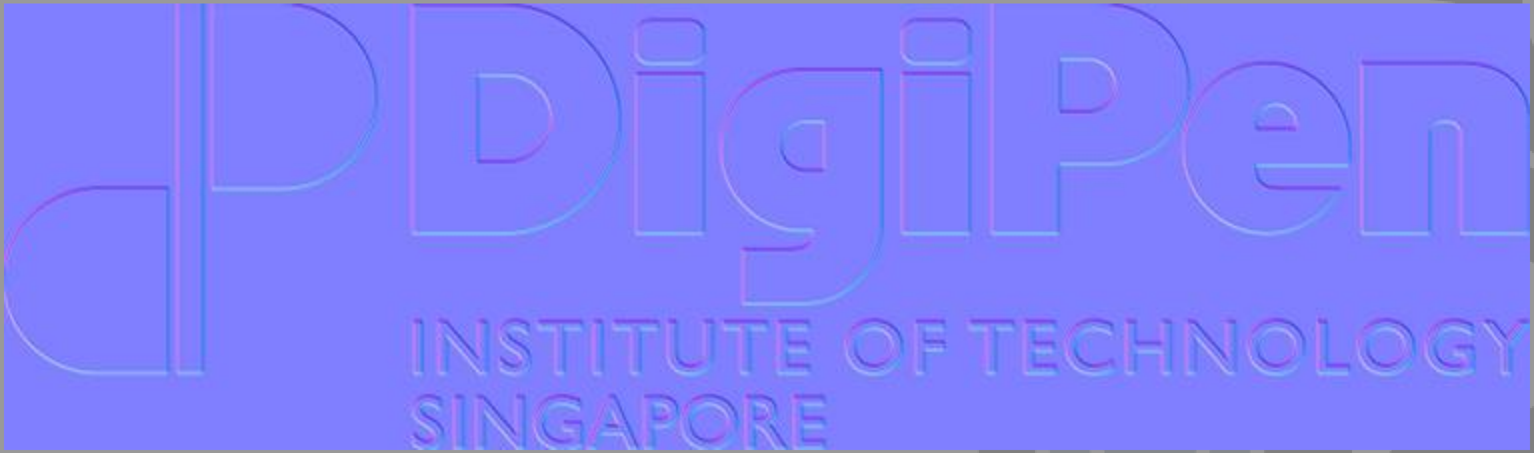
- More on Lights
  - Code can be adjusted to support a MAX amount of light objects
  - Lights with diffuse and specular props can have an attenuation factor
    - » The light effect will diminish with the distance from the light source

# Dynamic Lights

## Basic Demo

# Dynamic Lights – Normal Maps

DigiPen
INSTITUTE OF TECHNOLOGY
SINGAPORE



# Dynamic Lights – Normal Maps

- A normal map is a texture with rgb values expressed as xyz
- Think of each pixel as a normal vector
- A normal map texture is a mask layer of the original texture

# Dynamic Lights – Normal Maps

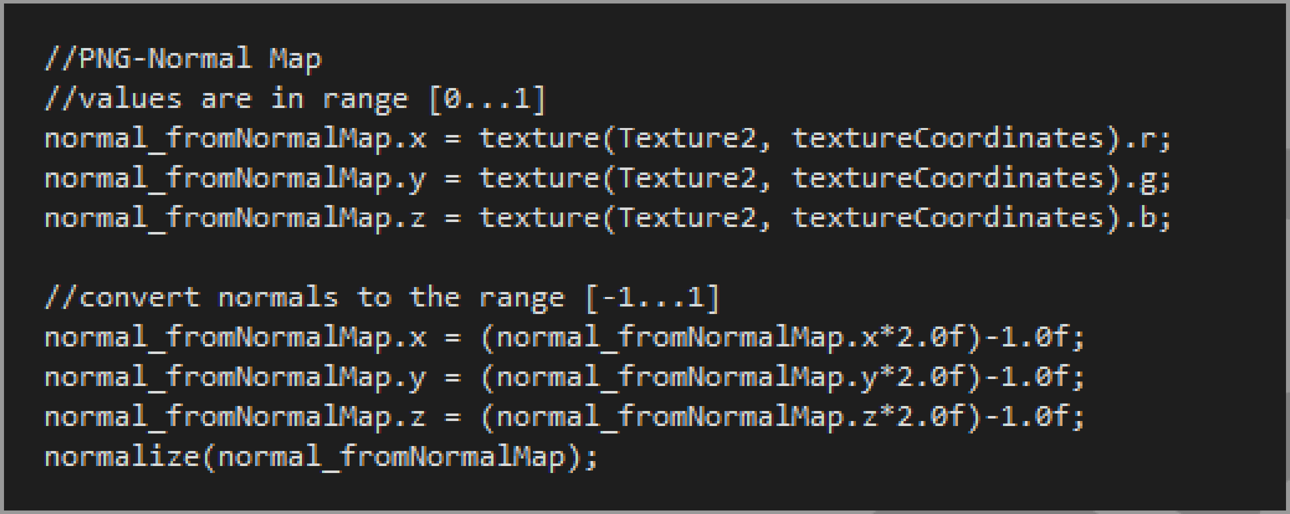
- When a light object is applied, the intensity of the colors are determined by replacing the *normal* with *normal_fromNormalMap* values
- Each normal that used to be (0.0f,0.0f,1.0f), in our previous fragment shader code, is now replaced with an (x,y,z) value read from the normal map texture

# Dynamic Lights – Normal Maps

- More than one method to apply normal map
- The one that we found easy to grasp is:
  - Bind 2 textures per fragment shader
  - In C++ code
    - » glActiveTexture(GL_TEXTURE0);
    - » glActiveTexture(GL_TEXTURE1);
  - In the fragment/pixel shader code
    - » uniform sampler2D Texture1;
    - » uniform sampler2D Texture2;

# Dynamic Lights – Normal Maps

- Sample shader code, reading a normal, per fragment, from the .PNG normal map texture (check Nvidia reference)



```
//PNG-Normal Map
//values are in range [0...1]
normal_fromNormalMap.x = texture(Texture2, textureCoordinates).r;
normal_fromNormalMap.y = texture(Texture2, textureCoordinates).g;
normal_fromNormalMap.z = texture(Texture2, textureCoordinates).b;

//convert normals to the range [-1...1]
normal_fromNormalMap.x = (normal_fromNormalMap.x*2.0f)-1.0f;
normal_fromNormalMap.y = (normal_fromNormalMap.y*2.0f)-1.0f;
normal_fromNormalMap.z = (normal_fromNormalMap.z*2.0f)-1.0f;
normalize(normal_fromNormalMap);
```

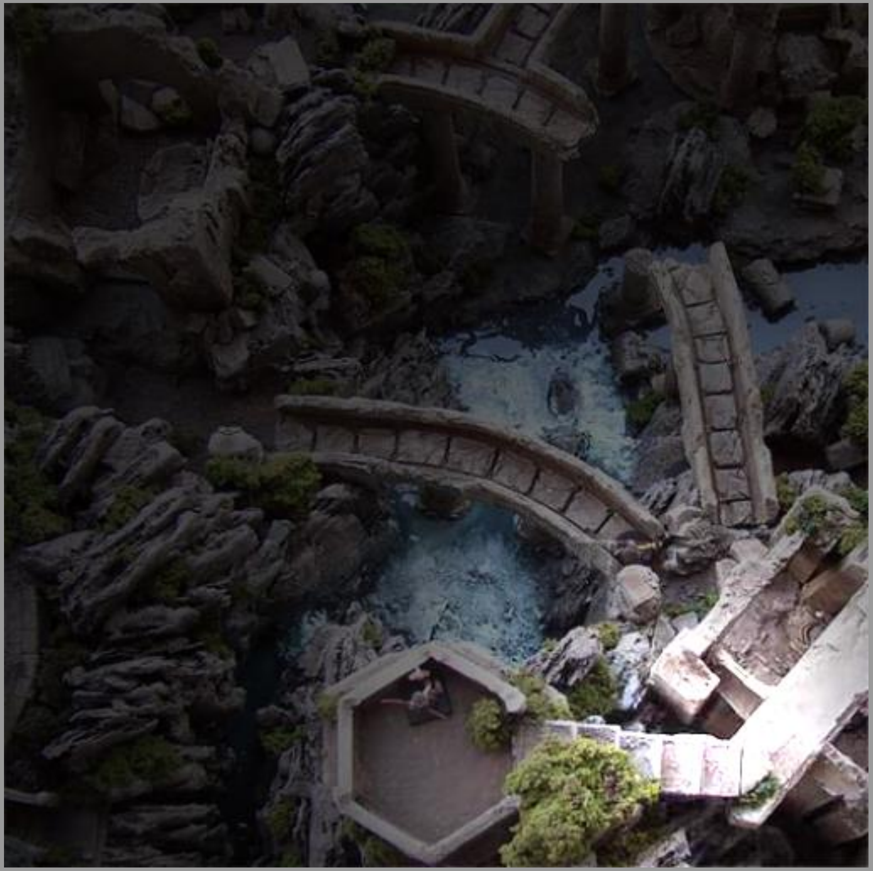
# Dynamic Lights – Normal Maps

- Loading a normal map in .dds format
  - Normal maps are compressed in different ways to minimize visual artifacts

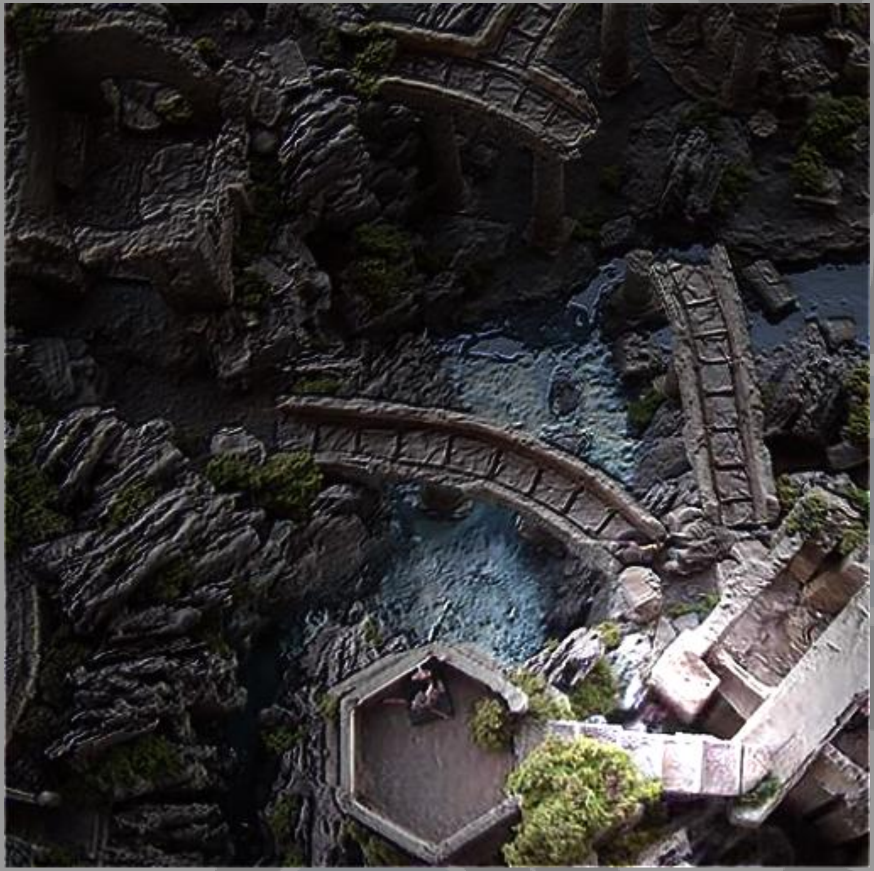
# Dynamic Lights – Normal Maps

- Loading a normal map in .dds format
  - Our method:
    - Convert a “*texture.png*” file to “*texture_nm.png*” file using a tool
      - e.g. online tool at “*http://www.smart-page.net/smartnormal/*”
    - Convert a “*texture_nm.png*” file to “*texture_nm.dds*” file using a tool
      - e.g. using Gimp2.8, export the normal map as “*DXT5nm*”
      - In the fragment shader compute the *z* value out of “*texture(Texture2, textureCoordinates).w*” and “*texture(Texture2, textureCoordinates).y*”, using Pythagoras theorem!

# Dynamic Lights – Normal Maps



With Normal Map



# Dynamic Lights – Normal Maps

Basic Demo

# References

- learnopengl.com
- https://www.nvidia.com/object/real-time-normal-map-dxt-compression.html

# Framebuffers

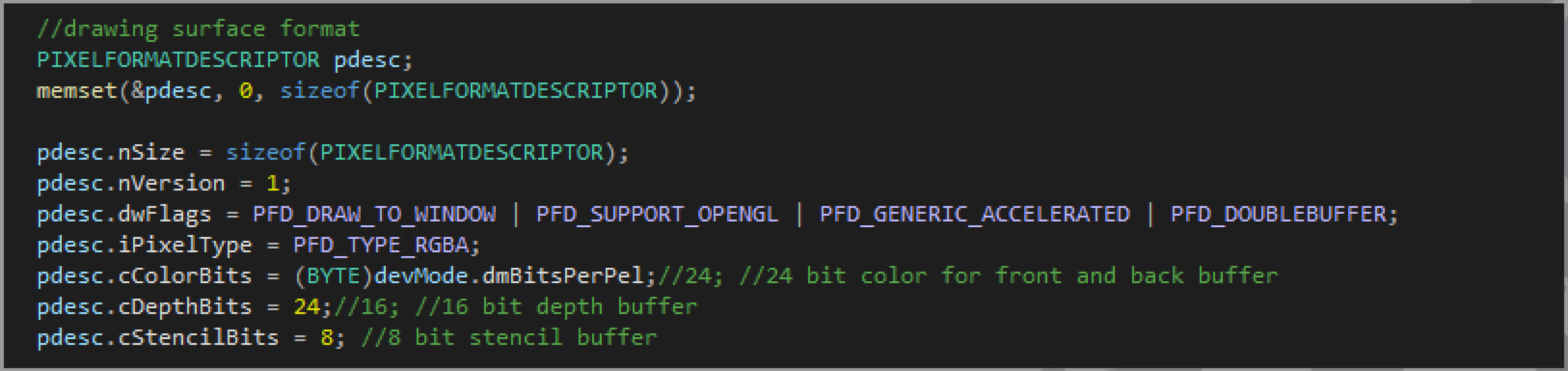
# Framebuffers

- By default, when you render your scene, you will be using the default frame buffer.
- A frame buffer is the buffer and sub-buffers holding all the rendering data with all their different states.
- Calling “glDrawElements” or “glDrawArrays” will draw to the current bound framebuffer.

# Framebuffers

- Default framebuffer is setup when setting up OpenGL with the Window's application.

- In example:



```
//drawing surface format
PIXELFORMATDESCRIPTOR pdesc;
memset(&pdesc, 0, sizeof(PIXELFORMATDESCRIPTOR));

pdesc.nSize = sizeof(PIXELFORMATDESCRIPTOR);
pdesc.nVersion = 1;
pdesc.dwFlags = PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_GENERIC_ACCELERATED | PFD_DOUBLEBUFFER;
pdesc.iPixelType = PFD_TYPE_RGBA;
pdesc.cColorBits = (BYTE)devMode.dmBitsPerPel;//24; //24 bit color for front and back buffer
pdesc.cDepthBits = 24;//16; //16 bit depth buffer
pdesc.cStencilBits = 8; //8 bit stencil buffer
```

# Framebuffers

- Using Additional Framebuffers.
  - Create your Framebuffer class
    - – Wraps a framebuffer data
  - Framebuffer data
    - – Rendering type: i.e. framebuffer used as a texture
    - – Applied post processing shader’s effect

# Framebuffers

- Basic usage of additional framebuffer as a texture
    - Steps in a graphics loop:
        - glBindFramebuffer(GL_FRAMEBUFFER, m_FBO);
        - //Render your game scene (as you did before)
        - glBindFramebuffer(GL_FRAMEBUFFER, 0);//back to default
        - glClearColor(0.5f, 0.5f, 0.5f, 1.0f);//example
        - glClear(GL_COLOR_BUFFER_BIT);
        - //Draw 1 quad and bind the framebuffer texture to it
        - //Swap your buffer. Must be the last step.

# Framebuffers

Sample Demo

# References

- learnopengl.com

# Questions?