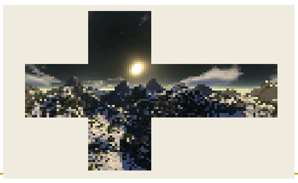
Cube mapping in GLSL

Overview

- A environment mapping technique to simulate a shiny object reflecting its surrounding environment
- Based on the assumption that the object's environment is infinitely distant from the object
- A chrome-like appearance is achieved by mapping the object with a cube map texture that encodes the object's environment

Cube map texture

- A cube map consists of SIX square texture images corresponding to the six faces of a cube
- Form an omni-directional image of an environment



Draw environment

- Environment mapping is typically implemented in at least two passes:
  - Pass 1: Draw the environment
  - Pass 2: Draw the cube mapped object
- To draw environment
  - Load an object to model the environment, such as a cube, a sphere
  - By Imagining the surface of the object is distant, we can safely regard the view position is at the coordinate origin and use the position of the vertex as the view direction

Draw environment

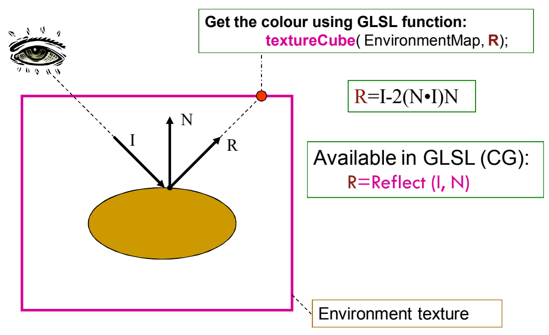
Vertex Shader

```
uniform vec4 fvEyePosition;
varying vec3 vDir;

void main(void)
{
    vDir = gl_Vertex.xyz;
    vec3 Pos2beViewed = fvEyePosition.xyz + vDir ;
    gl_Position = gl_ModelViewProjectionMatrix * vec4(Pos2beViewed, 1.0);
}
```

Pixel Shader

```
uniform samplerCube skyBox;
varying vec3 vDir;
void main(void)
{
  gl_FragColor = textureCube( skyBox, vDir);
}
```

Reflective environment mapping



Vertex shader

- Must pass information to fragment shader for computing the reflection direction
- Two vectors are needed:

```
varying vec3 vNormal;  //normal vector
varying vec3 vView; //view direction
```

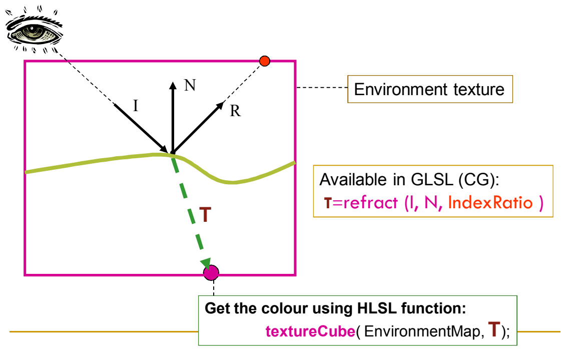
- Compute normal and view vector:

```
void main(void)
{
   … …
   vNormal = gl_Normal;
   vView = view_position.xyz – Pos;
   … …
}
```

Vertex Shader

```
uniform samplerCube Environment;
varying vec3 vNormal;
varying vec3 vView;
… …
void main( ) {
    … …
    vec3 normal = normalize(vNormal);
    // Find the reflection
    vec3 reflVec =  reflect(-vView, normal);
    vec3 reflCol = textureCube( Environment, reflVec).xyz;
    … …
}
```

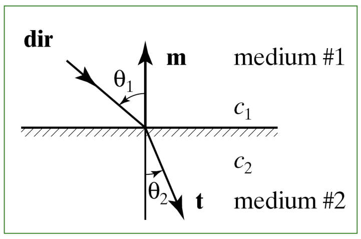
Refractive Environment mapping



Available in GLSL (CG):
T=refract (I, N, IndexRatio)

Get the colour using HLSL function:
textureCube(EnvironmentMap, T);

The refraction of Light: The Snell's law



$$\frac{\sin(\theta_2)}{c_2} = \frac{\sin(\theta_1)}{c_1}$$

Refraction indices of some common materials

Material	Index of refraction
Vacuum	1.0
air	1.0003
Water	1.3333
Glass	1.5
Plastic	1.5

Vertex shader

- Similar to the implementation of the reflection effect
- Must pass information to pixel shader for computing the refraction direction
- Two vectors are needed:

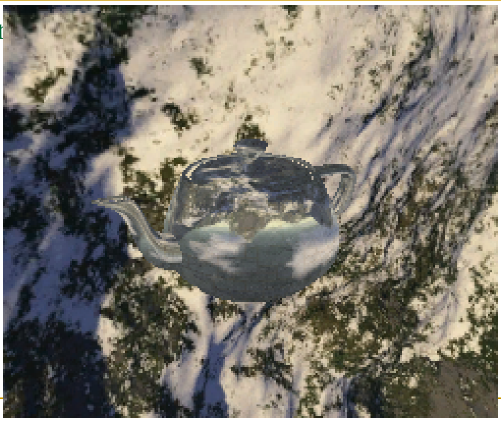
```
varying vec3 vNormal;  //normal vector
varying vec3 vView; //view direction
```

- Compute the above variables in main( )

Pixel Shader

```
//Similar input to the implementation of reflective effect
… …
void main( ) {
    … …
    vec3 normal = normalize(vNormal);
    // Find the refraction
    vec3 refrVec =  refract(-vView, normal, refractRatio);
    vec3 reflCol = textureCube( Environment, refrVec).xyz;
    … …
}
```

I



Shader Based Animation

- Fragment Based Techniques
- Vertex Based Animation

Introduction

- Animation is to simulate the behaviour of objects over time
  - Position, orientation, and colour of an object are considered as functions of time
- Must keep a track of time
- Create animation on the GPU is much more efficient than on the CPU

Fragment Based Animation: Idea

- Each texel is associated with a vertex and change the texture coordinate associated with a vertex will change the colour value associated with the vertex
- By just altering the sampling texture coordinates, it is possible to animate certain dynamic behaviours of an object using a texture
  - Depending on the use of a texture, the animation can be based on altering:
    - Colour
    - Position
    - Velocity
    - Normal
    - … …

Image Transformation

- Create a timer
- Specify how the image will be transformed
  - Translation
  - Rotation
  - Scaling
  - … …

Example: Image Based Fire Animation

- Use THREE textures to specify:
  - Fire shape
  - Fire intensity
  - Noise
- Noise texture is used to alter fire shape and fire intensity

Example 1: Image Based Fire Animation —Vertex Shader

- Load a screen aligned rectangle and mapped with the three textures
- May need to sample different textures differently:

```
vec2 T0 = gl_MultiTexCoord0;
vec2 T1 = gl_MultiTexCoord0;
… …
```

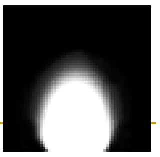
Example 1: Image Based Fire Animation — Pixel Shader : Control Fire Shape

- Introduce a timer

```
float time;
```

- Specify the overall shape of the fire using a texture

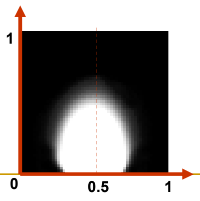
```
vec4 FireShape=tex2D( baseMap, T0);
```

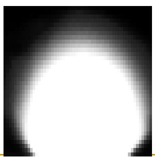


Example 1: Image Based Fire Animation — Pixel Shader : Control Fire Size

- Control the size of the fire shape by scaling the fire shape texture in the following way:

```
vec2 NewTexCoords = T0;
NewTexCoords.x = Scale.x*(T0.x-0.5) + 0.5;
NewTexCoords .y = Scale.y*T0.y;
```

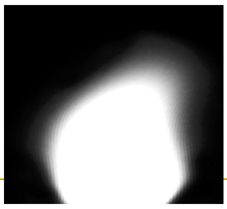




Example 1: Image Based Fire Animation — Pixel Shader : Control Fire Shape

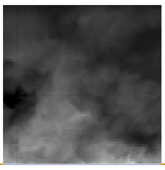
- Perturb fire shape along x-axis and y-axis:

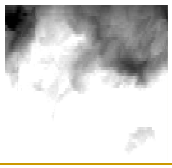
```
NewTexCoords.x += 0.1*T0.y*sin( freq.x *(T0.y + time) );

NewTexCoords.y += 0.1*T0.y*sin( freq.y * (T0.x + time) )*sin(freq.y*T0.x );
```



Example 1: Image Based Fire Animation — Pixel Shader : Add some noise

```
//shooting the texel upward:
T1.y += speed * time;
NewTexCoords += texture2D(fire_noise, T1).xy
                        * distortion_amount;
```





Example 1: Image Based Fire Animation — Pixel Shader : Control Fire Intensity

- Modify fire intensity using the noise and shape textures

```
vec4 Basefire= texture2D ( baseMap, NewTexCoords );
return Noise*FireShape*Basefire;
```