XIAMEN UNIVERSITY

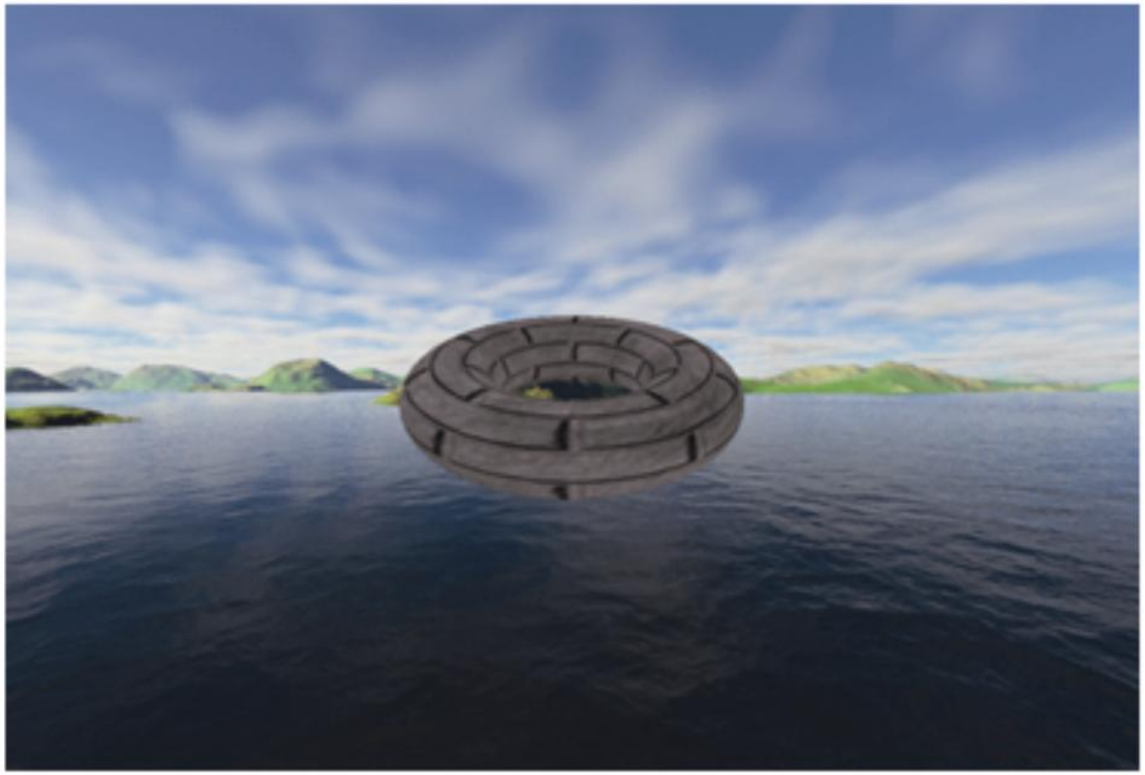
COMPUTER GRAPHICS

第九章 天空和背景

陈中贵
厦门大学信息学院
http://graphics.xmu.edu.cn

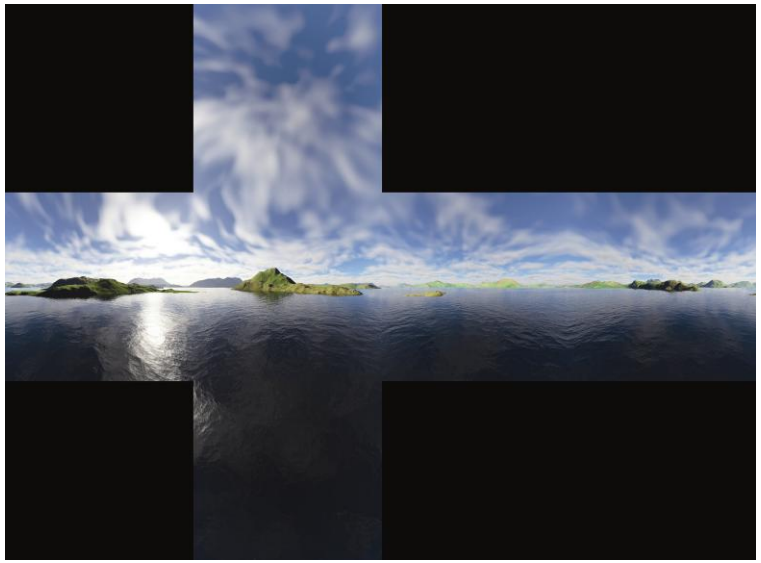
天空和背景

- 远处的大型物体，如云、群山或太阳
- 将这些对象作为单个模型添加到场景中可能会产生高到无法承受的性能成本
- **天空盒**或**穹顶**技术用来生成逼真的地平线景观



天空盒

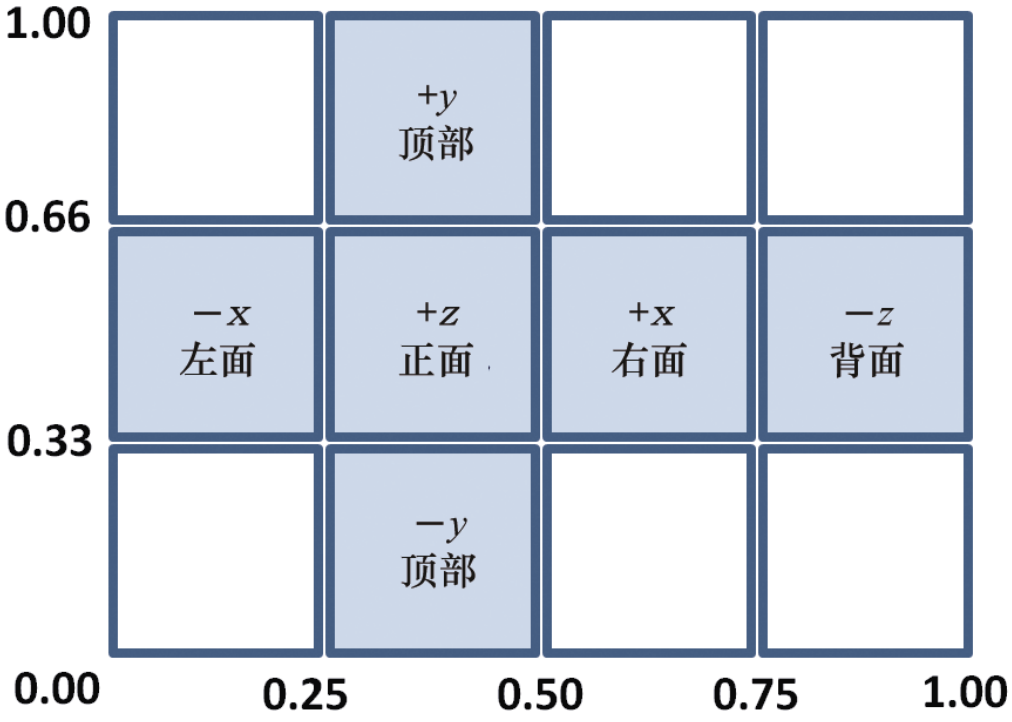
- （1）实例化一个立方体对象；
- （2）将立方体的纹理设置为所需的环境；
- （3）将立方体围绕相机放置。



6 面天空盒纹理立方体贴图

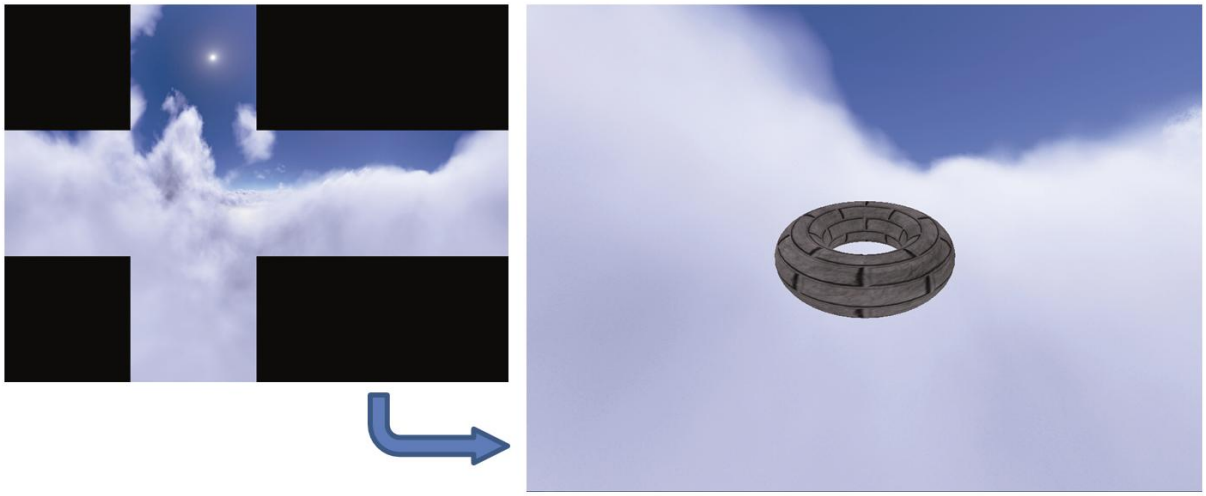
立方体贴图包裹相机

天空盒纹理坐标



如何让天空盒看起来“距离很远”？

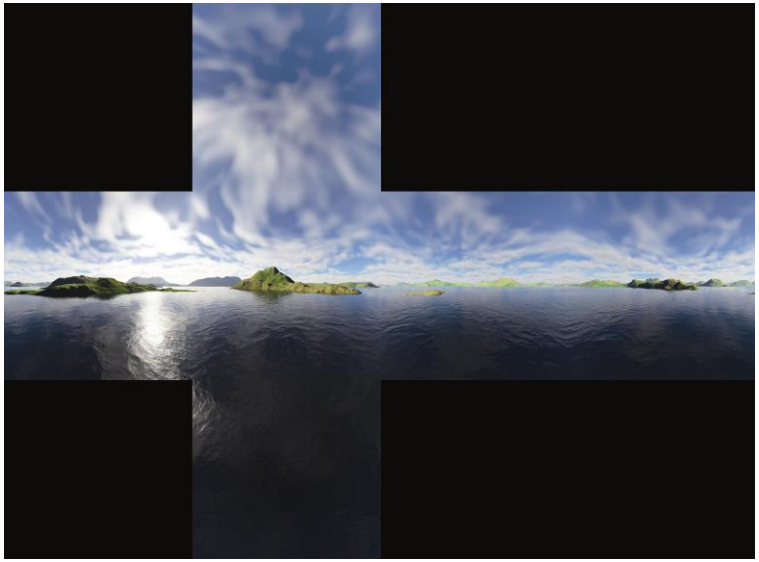
- 禁用深度测试并先渲染天空盒
  - 通过在禁用深度测试的情况下先绘制天空盒，深度缓冲区的值仍将全设为 1.0（即最远距离）。
  - 在渲染场景中的其他对象时重新启用深度测试
- 使天空盒随相机移动（如果相机需要移动）。

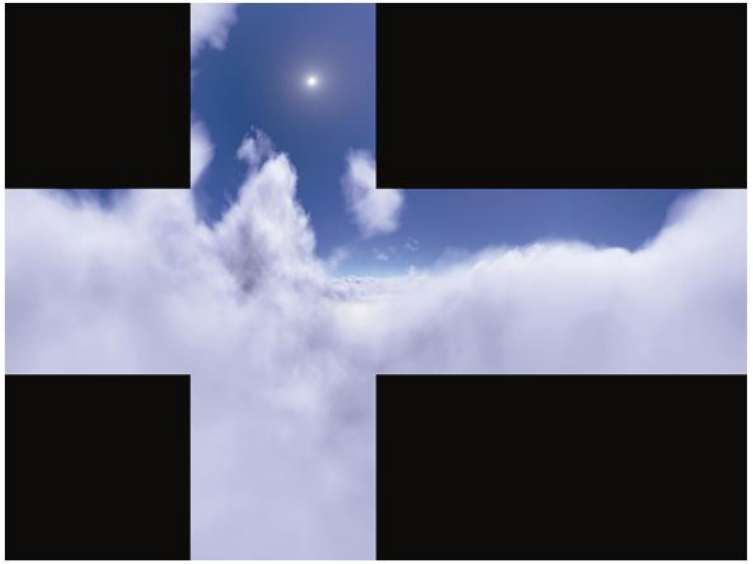


从天空盒内部查看场景

如何构建纹理立方体贴图?

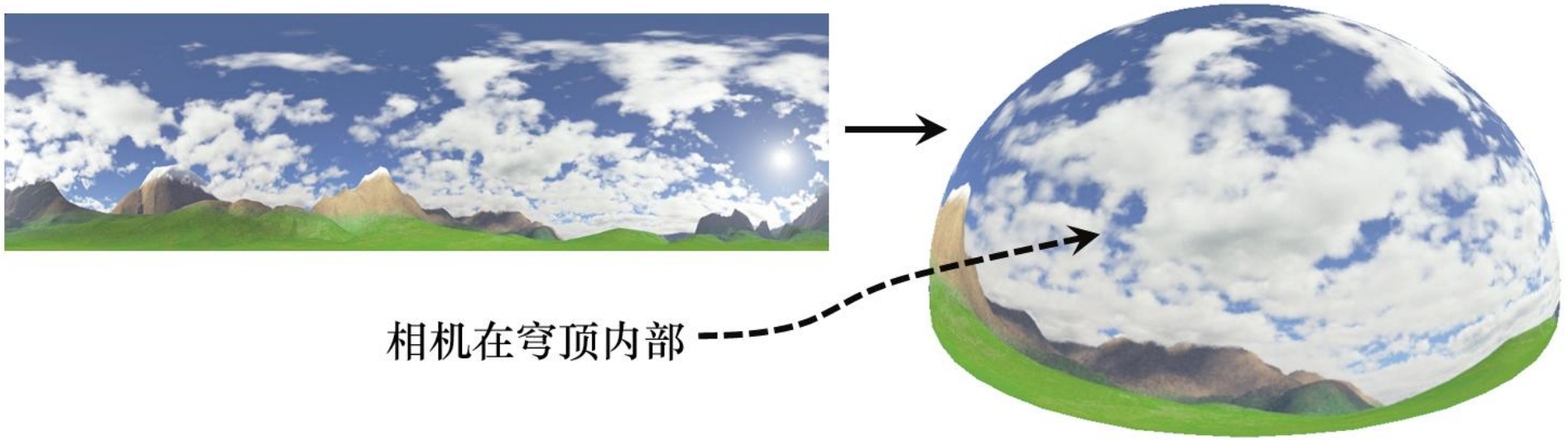
- 避免在立方体面交汇点处的“接缝”
- 工具：Terragen、 Autodesk 3Ds Max、 Blender 和 Adobe Photoshop
- 一些网站提供现成的立方体贴图





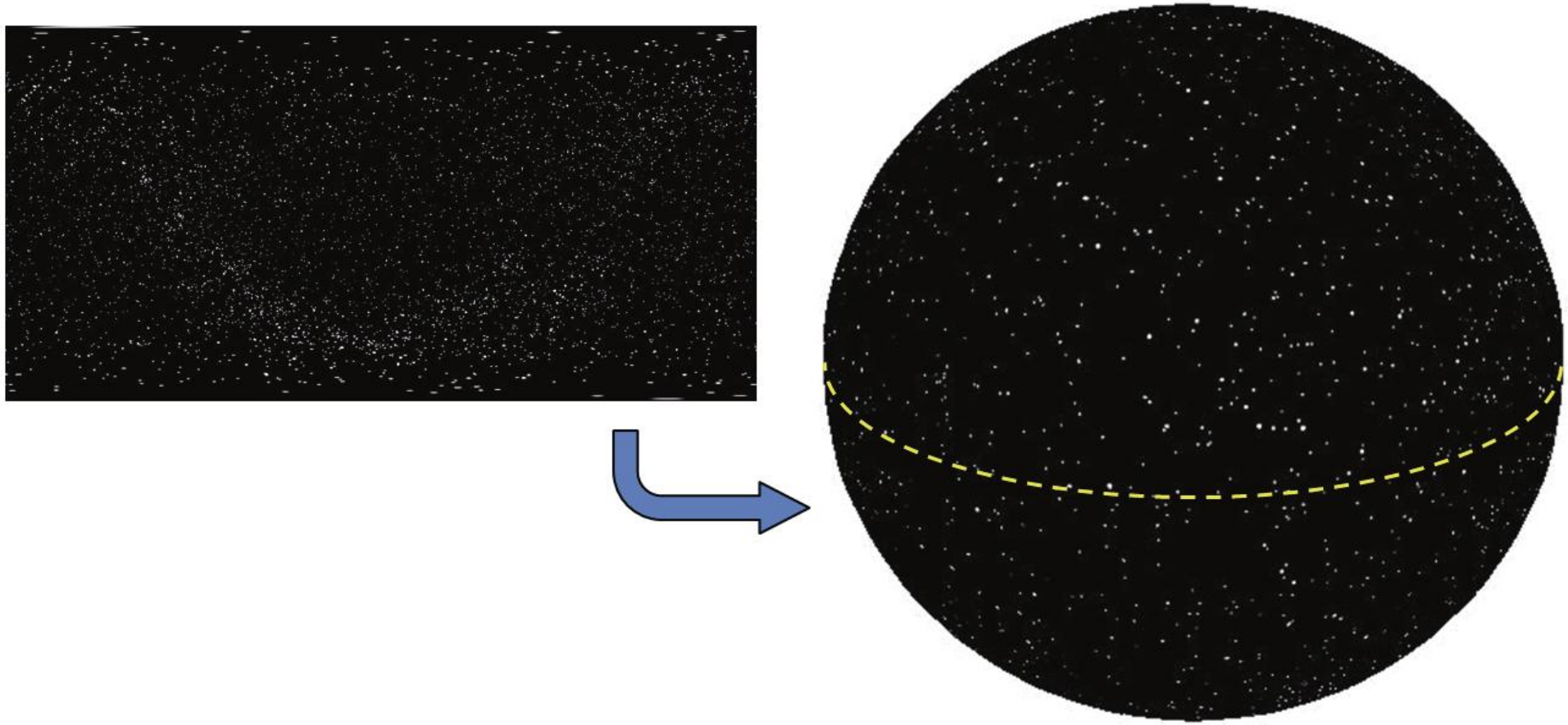
穹顶

- 使用带纹理的球体（或半球体）
- 不易受到畸变和接缝的影响（极点处的球形畸变）
- 球体或穹顶模型比立方体模型更复杂，穹顶有更多的顶点



穹顶

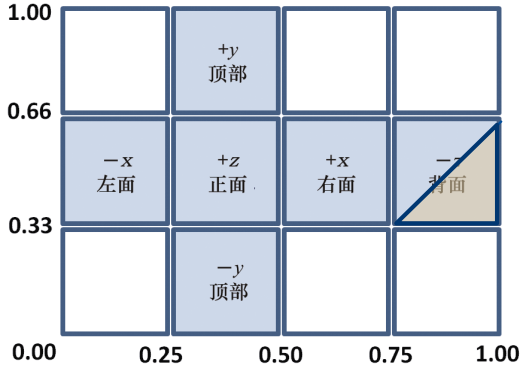
- 使用球体的星空穹顶



实现天空盒

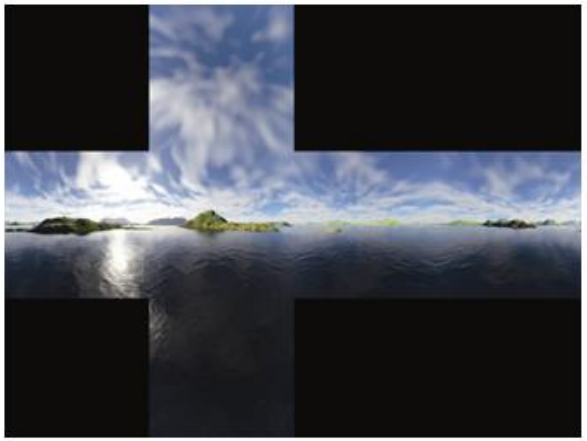
```
void display(GLFWwindow* window, double currentTime)  {
    . . .
    //  Draw the skybox first.  The M matrix places the skybox at the camera location
    mMat = glm::translate(cameraX, cameraY, cameraZ);
    . . .
    //  activate the skybox texture
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, skyboxTexture);
    glEnable(GL_CULL_FACE);
    glFrontFace(GL_CCW);   // cube is CW, but we are viewing its interior
    glDisable(GL_DEPTH_TEST);
    glDrawArrays(GL_TRIANGLES, 0, 36);   // draw skybox without depth testing
    glEnable(GL_DEPTH_TEST);

    //  now draw desired scene objects as before
    . . .
    glDrawElements( . . . );   // as before for scene objects
}
```

(continued)

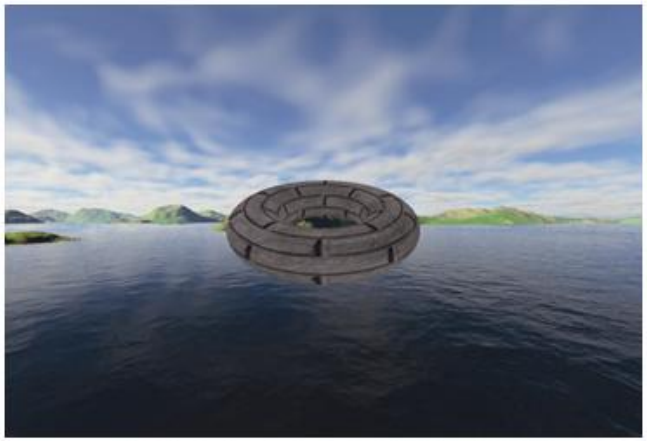



```
void setupVertices(void)  {
    //  cube vertices defined same as before
    //  cube texture coordinates for the skybox:
    float cubeTextureCoord[72] = {
        1.00f, 0.66f, 1.00f, 0.33f, 0.75f, 0.33f,     // back face lower right triangle
        0.75f, 0.33f, 0.75f, 0.66f, 1.00f, 0.66f,     // back face upper left
        0.75f, 0.33f, 0.50f, 0.33f, 0.75f, 0.66f,     // right face lower right
        0.50f, 0.33f, 0.50f, 0.66f, 0.75f, 0.66f,     // right face upper left
        0.50f, 0.33f, 0.25f, 0.33f, 0.50f, 0.66f,     // front face lower right
        0.25f, 0.33f, 0.25f, 0.66f, 0.50f, 0.66f,     // front face upper left
        0.25f, 0.33f, 0.00f, 0.33f, 0.25f, 0.66f,     // left face lower right
        0.00f, 0.33f, 0.00f, 0.66f, 0.25f, 0.66f,     // left face upper left
        0.25f, 0.33f, 0.50f, 0.33f, 0.50f, 0.00f,     // bottom face upper right
        0.50f, 0.00f, 0.25f, 0.00f, 0.25f, 0.33f,     // bottom face lower left
        0.25f, 1.00f, 0.50f, 1.00f, 0.50f, 0.66f,     // top face upper right
        0.50f, 0.66f, 0.25f, 0.66f, 0.25f, 1.00f      // top face lower left
        };
    // set up buffers for cube and scene objects as usual
}
//  modules for loading shaders, textures, etc. as before
```

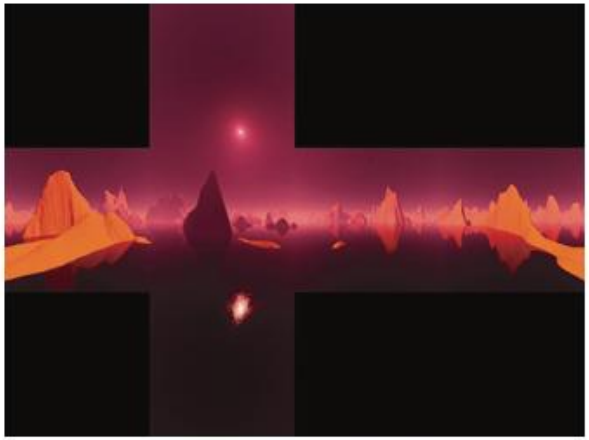
简单天空盒渲染结果



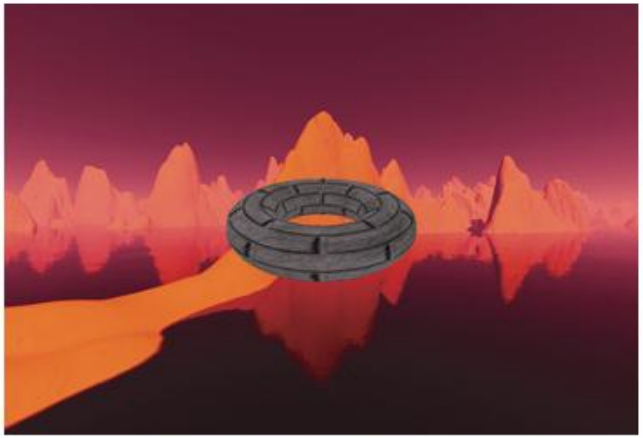
立方体贴图纹理（1）



纹理（1）天空盒中所渲染的场景



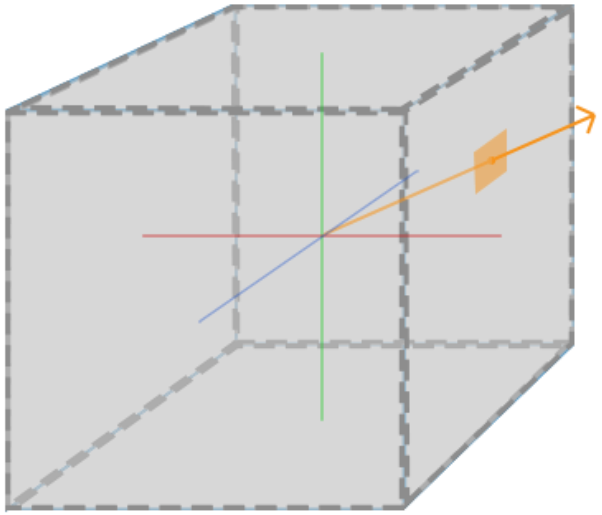
立方体贴图纹理（2）



纹理（2）天空盒中所渲染的场景

使用 OpenGL 立方体贴图

- 将多个纹理组合起来映射到一个单一纹理，它就是**立方体贴图(Cube Map)**
- 含6个2D纹理，这每个2D纹理是一个立方体（cube）的一个面，也就是说它是一个有贴图的立方体
- 使用方向向量对它们索引和采样
- 方向向量的大小无关紧要。一旦提供了方向，OpenGL就会获取方向向量触碰到立方体表面上的相应的纹理像素（texel）



OpenGL Cube Maps

```
void init(GLFWwindow* window)  {
    renderingProgramCubeMap = Utils::createShaderProgram("vertCShader.glsl",
                                                          "fragCShader.glsl");
    . . .
    skyboxTexture = Utils::loadCubeMap("cubeMap");
    glEnable(GL_TEXTURE_CUBE_MAP_SEAMLESS);
}
void display(GLFWwindow* window, double currentTime)  {
    //  draw cube map first, using its own rendering program
    glUseProgram(renderingProgramCubeMap);

    . . .
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_CUBE_MAP, skyboxTexture);
    . . .
    glEnable(GL_CULL_FACE);
    glFrontFace(GL_CCW);
    glDisable(GL_DEPTH_TEST);
    glDrawArrays(GL_TRIANGLES, 0, 36);
    glEnable(GL_DEPTH_TEST);
    . . .  // draw remainder of the scene
}
```

(continued)

```
GLuint Utils::loadCubeMap(const char *mapDir)  {
    GLuint textureRef;

    // assumes the six file names are xp, xn, yp, yn, zp, and zn, and all are JPG format
    string xp = mapDir; xp = xp + "/xp.jpg";
    string xn = mapDir; xn = xn + "/xn.jpg";
    string yp = mapDir; yp = yp + "/yp.jpg";
    string yn = mapDir; yn = yn + "/yn.jpg";
    string zp = mapDir; zp = zp + "/zp.jpg";
    string zn = mapDir; zn = zn + "/zn.jpg";

    textureRef = SOIL_load_OGL_cubemap(
            xp.c_str(), xn.c_str(), yp.c_str(), yn.c_str(), zp.c_str(), zn.c_str(),
            SOIL_LOAD_AUTO, SOIL_CREATE_NEW_ID, SOIL_FLAG_MIPMAPS);

    if (textureRef == 0) cout << "didnt find cube map image file" << endl;

    // to help reduce seams:
    gl.glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S,
                                                    GL_CLAMP_TO_EDGE);
    gl.glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T,
                                                    GL_CLAMP_TO_EDGE);
    gl.glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R,
                                                    GL_CLAMP_TO_EDGE);
    return textureID;
}
```

Vertex shader

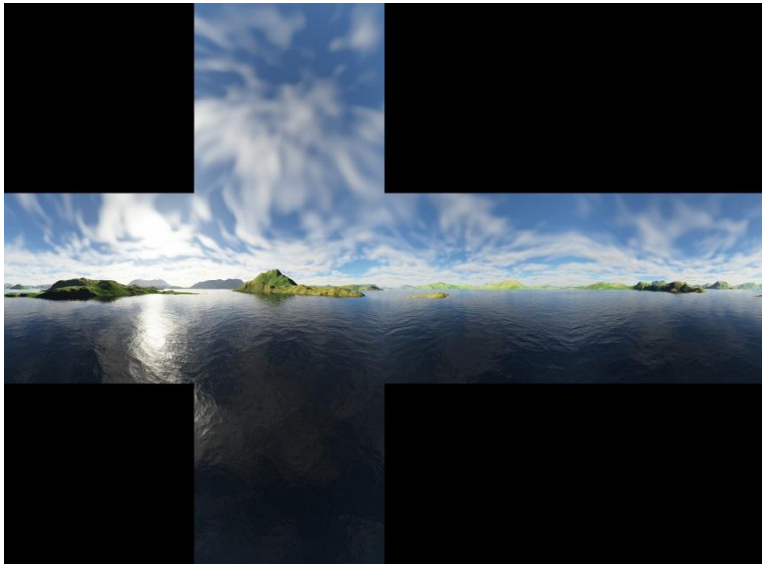
```
. . .
// (tc and position are vertex attributes as before)
. . .
layout (binding = 0) uniform samplerCube samp;

void main(void)
{   tc = position;       // texture coordinates are simply the vertex coordinates
    mat4 vrot_matrix = mat4(mat3(v_matrix));    // removes translation from view matrix
    gl_Position = p_matrix * vrot_matrix * vec4(position, 1.0);
}
```

Fragment shader

```
. . .
layout (binding = 0) uniform samplerCube samp;

void main(void)
{   fragColor = texture(samp,tc);
}
```

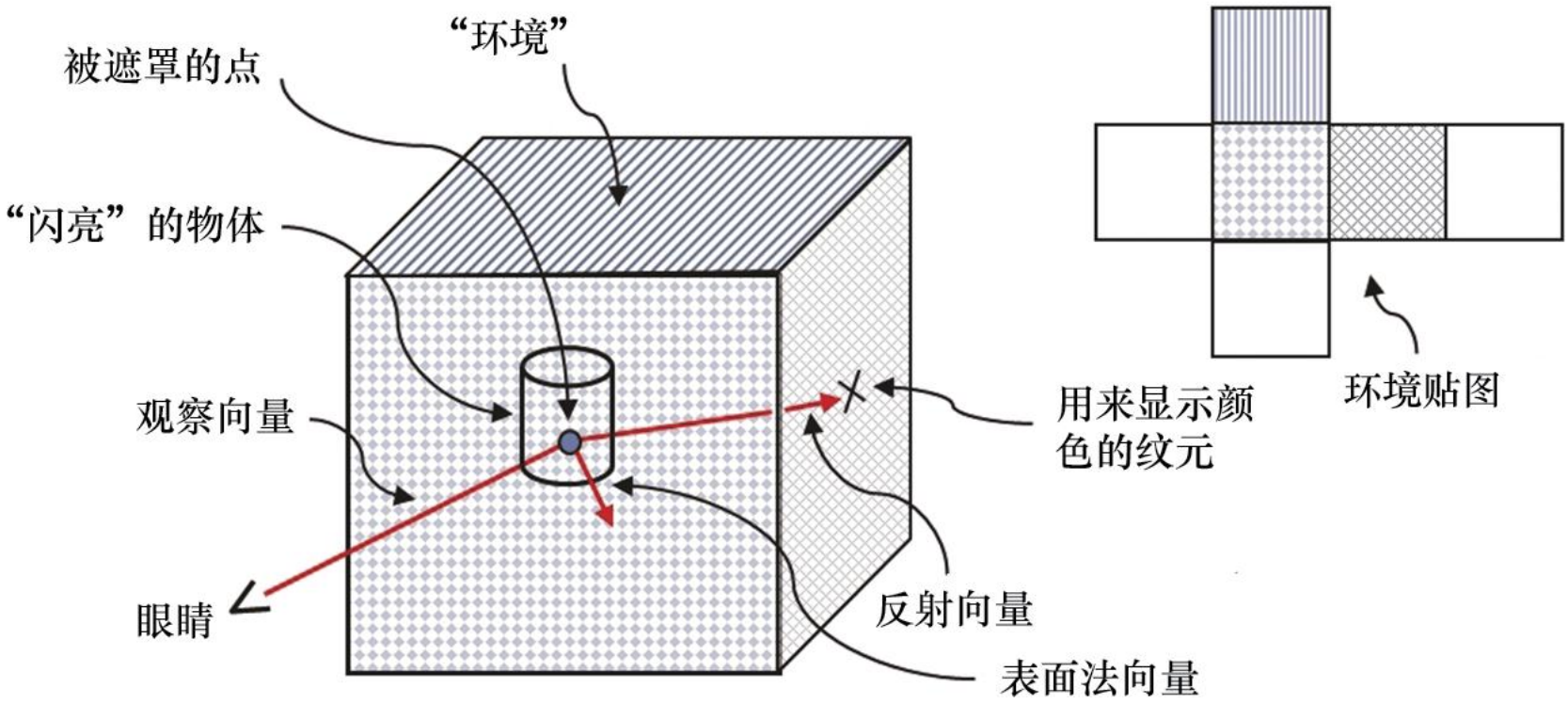
环境贴图

- 非常闪亮的物体 反射出周围物体的镜像
- ADS 光照模型并没有提供模拟这种效果的方法



环境贴图

- 通过立方体贴图模拟反射表面
- 使用观察向量和法向量组合计算反射向量



环境贴图

- 在 init()函数中
  - 创建环面的法向量缓冲区
  - 不再需要环面的纹理坐标缓冲区。
- 在 display()函数中：
  - 创建用于变换法向量的矩阵
  - 激活环面法向量缓冲区
  - 激活纹理立方体贴图为环面的纹理
- 在顶点着色器中：
  - 将法向量和 norm_matrix 相乘
  - 输出变换的顶点和法向量以备计算反射向量
- 在片段着色器中：
  - 以与第 7 章中相似的方式计算反射向量
  - 从纹理（现在是立方体贴图）检索输出颜色，使用反射向量而非纹理坐标进行查找

```
void display(GLFWwindow* window, double currentTime )  {
   . . .
   nLoc = glGetUniformLocation(renderingProgram, "norm_matrix");
   . . .
   invTrMat = glm::transpose(glm::inverse(mvMat));
   glUniformMatrix4fv(nLoc, 1, GL_FALSE, glm::value_ptr(invTrMat));
   . . .
   //  we need to activate the torus normals buffer:
   glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
   glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
   glEnableVertexAttribArray(1);
   //  the torus texture is now the cube map
   glActiveTexture(GL_TEXTURE0);
   glBindTexture(GL_TEXTURE_CUBE_MAP, skyboxTexture);
   //  drawing the torus is otherwise unchanged
   glClear(GL_DEPTH_BUFFER_BIT);
   glEnable(GL_CULL_FACE);
   glFrontFace(GL_CCW);
   glDepthFunc(GL_LEQUAL);
   glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[3]);
   glDrawElements(GL_TRIANGLES, numTorusIndices, GL_UNSIGNED_INT, 0);
}
```

Vertex shader

```
. . .
layout (location = 1) in vec3 normal;
out vec3 varyingNormal;
out vec3 varyingVertPos;
. . .
layout (binding = 0) uniform samplerCube tex_map;
void main(void)
{    varyingVertPos = (mv_matrix * vec4(position,1.0)).xyz;
     varyingNormal = (norm_matrix * vec4(normal,1.0)).xyz;
     gl_Position = proj_matrix * mv_matrix * vec4(position,1.0);
}
```

Fragment shader

```
. . .
in vec3 varyingNormal;
in vec3 varyingVertPos;
. . .
layout (binding = 0) uniform samplerCube tex_map;
void main(void)
{    vec3 r = reflect(normalize(-varyingVertPos), normalize(varyingNormal));
     fragColor = texture(tex_map, r);
}
```

环境贴图

- 即使没有任何 ADS 光照，也能让人感觉光经物体反射出来
- 环面的左下方似乎有一个镜面高光，因为立方体贴图中太阳在水中反射的倒影
- **局限性：**只能构建反射立方体贴图内容的对象。在场景中渲染的其他对象并不会出现在使用贴图模拟反射的对象中