Andy Jeong
Professor Sable
ECE462 Computer Graphics
5 May 2019

Environment Mapping using Cube-map

This project deals with implementation of environment mapping using cubemap structure, which is supported by WebGL, unlike spheremap structure. This technique, first proposed by Blinn and Newell in 1976, is a cheap way to create reflections on curved surfaces using texture mapping that is supported by the graphical hardware. The basic steps to implementation are as follows: first create a 2D environment map with 6 images—positive and negative in all x, y, and z directions,—compute the normal for each pixel on a reflective object, compute the reflection vector from eye position and face normal, compute an index into the environment texture using the reflection vector, and color the pixel using the corresponding texel.

This project uses a skybox—a large 6-sided cube surrounding the camera at the center, with each side textured with continuous images in all three dimensions—and uses reflection vectors from the light source to display a reflective texture on an object at the center of the cube. Before computing necessary vectors, the input 3D object file is parsed by vertex and face (index of vertices) elements to obtain geometric mesh structure. The object file produced from Blender software produces various components such as texture vertices (‘vt’), vertex normal (‘vn’) and parameter space vertices (‘vp’), but for simplicity and calculation of normals separately, only points (‘v’) and face (‘f’) values are used so that each per-face normal is calculated for every three vertices individually [2].

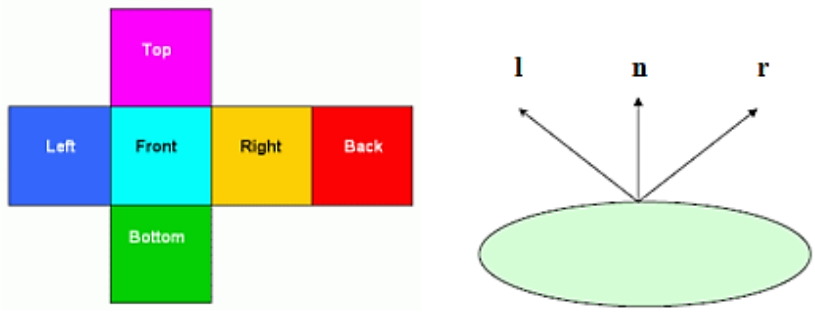

Figure 1. Cubemap and Vector Orientations

After parsing, three successive points are taken to produce a vector, and for three of these vectors, the surface normal is computed by cross product of face edges[1].

$$\text{Given vectors } (v1, v2, v3), \quad \text{vec3 normal} = \frac{(v2-v1)\times(v3-v1)}{\|(v2-v1)\times(v3-v1)\|}$$

Equation 1. surface normal calculations

Then each vertex normal, which is needed to compute its corresponding reflection vector, is computed by taking linear combination of face normal and normalizing, assuming each face has equal weights associated at the point. This allows for vertices, faces, and normals in hand to work with.

The points for skybox cubemap is defined hard-coded, for each of six sides (4 points per each) and sent to the array buffer, and indices for each triangle are specified and sent to the element array buffer. With the vertices defined, the image textures are placed by taking an image for each of x, y, and z dimension, and attaching to the s, t-coordinates for texture mapping. With the cubemap in place, the camera is then placed at the center of the map, and the model view matrix is computed by lookAt() function, which takes in position of the viewer (eye), the target to which the viewer is oriented (center), and directional vector pointing up (‘up’)—here, ‘eye’ and ‘up’ are translated according to the position of the camera, as we want it to be at the center, not anywhere else. The perspective matrix pMatrix is also computed from the input angle.

For final fragment colors on the reflective object, each lighting component(ambient, diffuse, specular) is multiplied to the skybox colors with the specified weights. Diffuse light weight is computed by taking maximum of dot product of light and normal vectors, and specular light weight is computed by raising the dot product of reflection and eye directional vectors to the power of desired shininess[4]. The positions for the three light source components are initially defined to be at (x,y,z) = (1,1,1).

There are a number of attributes that are allowed for interaction—shininess of the reflective object, perspective angle view, position of the eye, position of the light source, scale of the object in all three directions, and also selection of cubemap and the reflective object of user's interest. Some of the Blender-made .obj files are obtained from Florida State University's web resources[3]. The choice of map and object is drawn from individual 6-sided image map and object file, respectively, and other directional attributes are controlled by sliders in (x, y, z) directions. The image paths are named accordingly so that each image goes to proper side of the cube. The object file may yield an incomplete structure (i.e. cube) because it only takes into account v and f elements, and mesh is built from those only. The primary reason for adding controls for perspective angle and position of the eye is to understand how the cubemap is drawn. For instance, the teddy bear object is intentionally added as an option, so that by manipulating perspective angle, the object, which is larger than the view initially, can be made smaller by "zooming out," while the cubemap looks a bit distorted. This is not a fault; this occurs because by the very nature of drawing cubemap and placing the camera at the center, when the camera is pulled outward the texture mapping is also drawn with it. Thus, the image may look elongated or distorted, depending on the object structure. Also, variation in position of the eye

allows to view the cubemap from a distance and understand what the user actually sees initially, while maintaining the object at its original position.

References

[1] Angel, Edward S., and Dave Shreiner. Interactive Computer Graphics: a Top-down Approach with WebGL. Pearson, 2015. Pg. 384.

[2] *MartinReddy.net*, www.martinreddy.net/gfx/3d/OBJ.spec.

[3] "OBJ Files A 3D Object Format." OBJ Files - A 3D Object Format, people.sc.fsu.edu/~jburkardt/data/obj/obj.html.

[4] WebGL Example: Phong / Blinn Phong Shading, www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/code/WebGLShaderLightMat/ShaderLightMat.html.