Environment Mapping

Yuxiao Zhang
University of Southern California
Los Angeles, United States
yuxiaozh@usc.edu

Zhengli Yu
University of Southern California
Los Angeles, United States
zhengliy@usc.edu

Zichang Liu
University of Southern California
Los Angeles, United States
zichangl@usc.edu

***Abstract*—Environment mapping, or reflection mapping, is an image-based lighting technique for rendering the appearance of a reflective surface by means of a precomputed environment texture. Compared with classical ray tracing approach, this technique provides an efficient rendering for objects like mirror and mental.**

In this paper, we introduce an environment mapping algorithm for reflective surface in a 3D scene based on a cube map. How we improve this algorithm and apply different reflective materials will also be covered in the paper.

***Keywords*—environment mapping; cube mapping; reflection; texture**

I. Introduction

We started this project based on the code of previous homework of CSCI 580, 3D rendering and graphics course. We do not use any libraries or tools other than what we have in class. Before, we have implemented phong shading and uv textures. Based on these contents, we are going to analyze and implement the environment mapping, refraction, and mix of the textures.

By reading and understanding papers, articles, and books in corresponding fields, we are trying to improve the methods and using two different algorithms to make the cube mapping. Also, we spend some time on studying how to make refraction and how to mix different textures.

II. Cube Mapping

A. General Information

In this project, cube mapping is used as the environment mapping method. This kind of method projects the environment onto six sides of a cube and create six square textures to store the data of environment [1]. These six images represent the front, back, left, right, up and down of the environment respectively. An example cube map is shown in Figure 1.

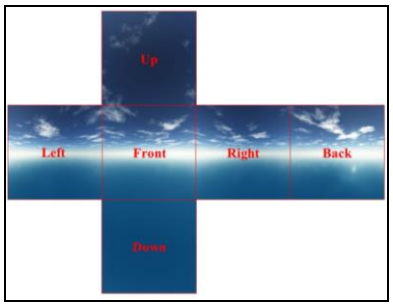


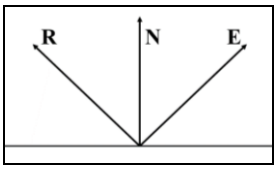
Figure 1 Cube Map

The cube map can be folded to become a box which warps up the whole world. Therefore, for any vector starting from the center point of the world, a unique corresponding point in the cube map can be found as the mapping point.

B. Reflection Vector and Corresponding Point

To calculate the reflection vector, a normal vector and an eyesight vector are needed. The normal vector can be obtained by interpolating and it has been calculated in shading period. By calculating the vector in image space, the eyesight vector will always be (0, 0, -1). Then the reflection vector can be calculated using the formula:

$$R = 2(N \cdot E) \cdot N - E$$



With the reflection vector R= (X, Y, Z), we assume that all reflection vectors starts form the center of the world, then the corresponding point in the cube map can be found. Firstly, the corresponding side of cube map (one of front, back, left, right, up and down) should be determined. The 3D coordinate system can be divided into three 2D coordinate systems (x-y, x-z and z-y). Each 2D coordinate system can be divided into 4 areas by $y = x$ and $y = -x$. Each area has its corresponding cube sides (Figure 2).

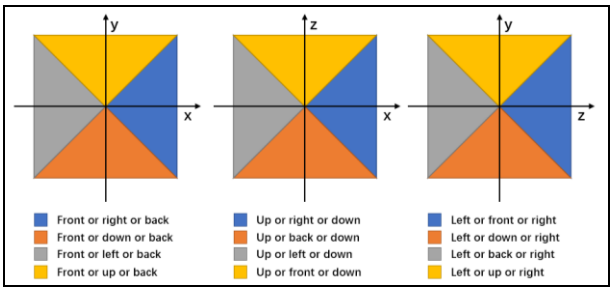


Figure 2 Division of 3D Coordinate System

By comparing the size of X, Y and Z of the vector R. The reflection vector can be linked with one cube side. Then, the reflection vector is projected to its relative cube side plane. The distance from the center to the plane is set to the maximum among X, Y and Z. The corresponding uv value of the reflection vector can be calculated according to the end point of the projected vector. Figure 3 shows how to calculate uv value of a vector pointing to the front.

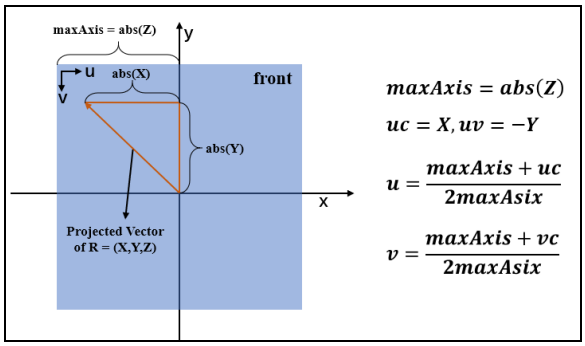


Figure 3 Convert xyz to uv

C. *Pseudo Code*

The following pseudo code shows general algorithm that maps a vector to a (u,v) value.

```
function convert_vector_to_uv(float x, float y, float z){
  if (x>0 && abs(x) >= abs(y) && abs(x) >= abs(z)) {
    maxAxis = abs(x); uc = -z; vc = -y;
    target = right;
  }else if (x<=0 && abs(x) >= abs(y) && abs(x) >= abs(z)) {
    maxAxis = abs(x); uc = z; vc = -y;
    target = left
  }else if (y>0 && abs(y) >= abs(x) && abs(y) >= abs(z)) {
    maxAxis = abs(y); uc = x; vc = z;
    target = up
  }else if (y<=0 && abs(y) >= abs(x) && abs(y) >= abs(z)) {
    maxAxis = abs(y); uc = x; vc = -z;
    target = down
  }else if (z>0 && abs(z) >= abs(x) && abs(z) >= abs(y)) {
    maxAxis = abs(z); uc = x; vc = -y;
    target = front
  }else if (z<=0 && abs(z) >= abs(x) && abs(z) >= abs(y)) {
    maxAxis = abs(z); uc = -x; vc = -y;
    target = back
  }
  u = 0.5 * (uc / maxAxis + 1);
  v = 0.5 * (vc / maxAxis + 1);
}
```

Figure 4 shows the result after applying the reflection to the teapot.

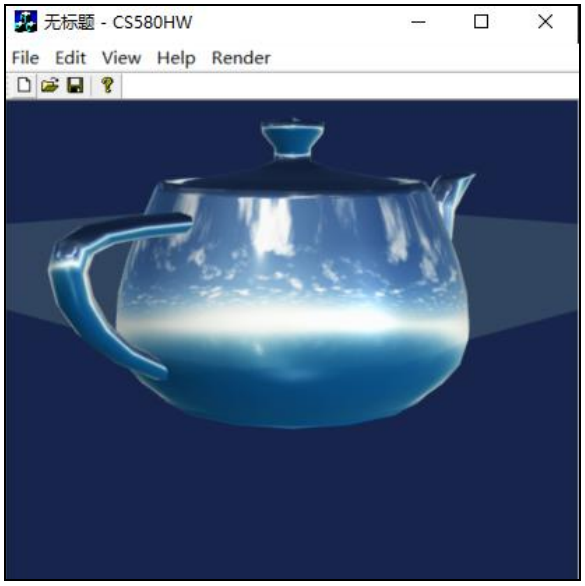


Figure 4 Teapot with reflection

D. *Reflection for Flat Surface*

This algorithm works well for curved surface. However, it cannot get an expected result for flat surface. In Figure 4, there is a reflective plane behind the teapot but all pixels on the plane have the same color. It is because we assume that all reflection vectors start from the center of world and eyesight vector $E = (0, 0, -1)$. For a flat surface where all points have same normal vector, all reflection vectors are same and point to the same point on the cube map. One of the solutions for this problem is to adjust the eyesight vector E for each point on the flat surface. The reflection vector is calculated in image space and camera is the center of image space. For any point P=(a,b,c) in image space, -P=(-a,-b,-c) can be used as eyesight vector for this point. Figure 5 shows the idea of this solution.

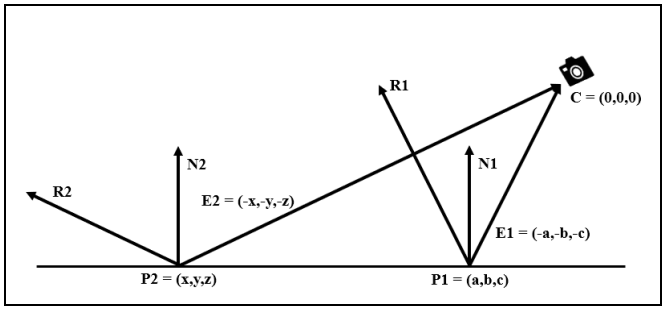


Figure 5 New eyesight vector for point in image space

Now the task is that, for each pixel on the screen, find its corresponding point in image space. Firstly, the inverse matrix of $X_{perspective\text{-}screen}$ is applied to the point on the screen space:

Given a P(x,y,z) in screen space, xRes and yRes are resolution of the screen:

$$PInPerspectiveSpace = (\frac{2x}{xRes} - 1,\ \frac{-2y}{yRes} - 1,\ \frac{z}{MAXINT})$$

Then the point in perspective space should be transferred into image space. Instead of using inverse matrix, the idea of parameter interpolation is used to get a fast calculation.

Given a P(x,y,z) in perspective space:

$$V_z' = \frac{z}{MAXINT - z}$$

$$PInImageSpace = (x * (V_z' + 1), y * (V_z' + 1), z * (V_z' + 1))$$

Figure 6 shows the result after applying the new algorithm:

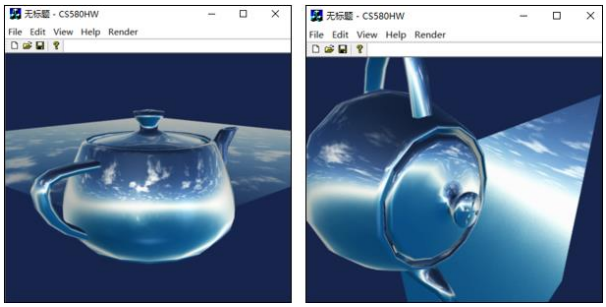


Figure 6 Reflection for flat surface

III. AN ALTERNATIVE WAY

A. Do Reflection with Pure Vector Calculation

After doing the reflection of the **E** Vector with **N**, the result **R** can be directly used as a coordinate to find UV point on one surface of the environment box. The **R** Vector is only considered to have the right direction start from reflection point P, but the length is variable. The actual UV point is the counterpoint of extended R and the surface. If there is a vector point at the actual point of UV then the coordinate can be decided. As demonstrates in Figure 7, the Red Vector is the final result to get. And it can be represented by a vector point to reflection point P by adding the extended **R** Vector. The reflection point P is known by the beginning of the calculation. Thus, the main work is to calculate the **R'** vector. (start from point P to the counterpoint on one surface)

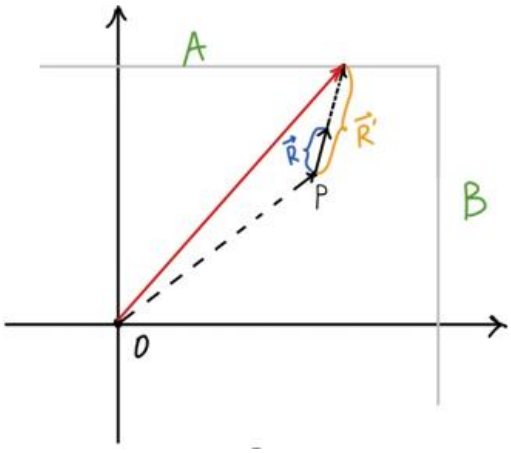


Figure 7 The Priority Setting of the Environment Box

When choosing to use an environment box to represent the reflection of surrounding objects, the volume of the box should be larger than the present object. And the relevant position of the object to the box also varies the result of reflection. Thus, before rendering the object, the box should be fixed in world space. Then the distance from start point O to each surface of the box is a constant value d. Take an example in a 2D coordinate, the vector point at the surface A can be (d,0). In the project, the Default distance from start Point O to each surface is 100.

B. The Calculation of R

Since the d is known, when executing a point of reflection, the position of point P is also a known value. Thus, the distance from the reflection point P to one surface can be confirmed. Once the distance from P to the surface is known, the vector **D** can be built up. Figure 8 demonstrates the calculation of **R'**.

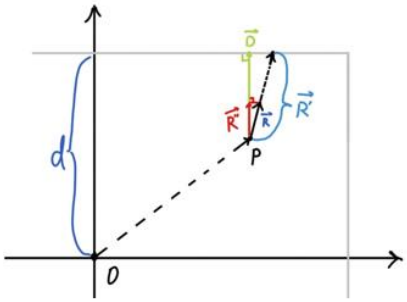


Figure 8 Calculation of R

Then an assistance vector **R"** is needed. The **R"** is the projection of vector **R** on vector **D**, the process is shown in the equation below.

$$R'' = Normalize(D) \cdot R$$

By using the Similar Triangle Property, the equation below can be confirmed

$$\frac{|R|}{|R''|} = \frac{|R'|}{|D|}$$

Through this equation, the Mol of **R'** can be ensured. Then the vector can be represented as below equation

$$R' = |R'| * Normalize(R)$$

Once the **R'** is confirmed, the red vector showed in Figure 7 can be represented as the sum of two vectors

$$V = R' + \overrightarrow{OP}$$

Now the coordinate of red Vector **V** can be used to get to actual UV value. With the pseudo-code shown in before.

C. *Property of the Pure Vector UV Fetch Method*

This method used the Similar Triangle Property to avoid the varies of the length of the R vector. It means whatever the **R** vector's length is, as long as its start point is in the environment box, the method will work. This method can deal with any kind of surface orientation, both flat and sphere like surface can be rendered perfectly.

IV. OTHER THAN REFLECTION

A. *Refraction*

For every incident ray, there is the refraction ray as well as the reflection ray. Refraction component will also affect the outcome of the RGB color. As we can see in figure 9, R is the refraction ray.

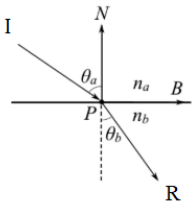


Figure 9

According to the derivation by Paul Heckbert [2], we can find the outcome vector of refraction by:

$$k = 1 - eta^2 * (1 - (N \cdot I)^2)$$
$$R = eta * I - (eta * (N \cdot I) + N * \sqrt{k}$$

Where eta is a ratio affected by the materials at two sides of the surface. N is the surface normal. I is the incident vector. R is the outcome refraction vector.
With this method, we can show the refraction result.

B. *Mix of the Textures and Mappings*

The cube mapping will overwrite the Ka, Ks, and Kd values which are used for calculating the RGB value of each pixel. To save original attributes of the object, or to show the textures of the object, we need to give each source a weighted value to show all of these elements.

$$Ka = \sum_{i=0}^{n} c_i * K_i$$
$$\sum c_i = 1$$

Where c is the weight of each source, and K is the value of it. Figure 10 and 11 are example outcomes of the combination of environment mapping and USC logo with different ratios.

Figure 10 c = 0.25 & 0.75

Figure 11 c = 0.5 & 0.5

V. CONCLUSION

In this paper, we mainly talk about how to calculate the reflection and how to find the target surface in a cube mapping. We mention general conditions and flat surfaces. After that, we also find out another algorithm to achieve the same goal. Also, we effort on extra topics we interested in, like refraction and multiple textures. Afterall, the final result may not be the perfect nor the best one, but we learned a lot from the study process, and felt the charm of graphics and rendering.

REFERENCES

[1] N. Greene, "Environment Mapping and Other Applications of World Projections," in *IEEE Computer Graphics and Applications*, vol. 6, no. 11, pp. 21-29, Nov. 1986.
doi: 10.1109/MCG.1986.276658W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4056759&isnumber=4056747

[2] HECKBERT, Paul S.; HANRAHAN, Pat. Beam tracing polygonal objects. In: *ACM SIGGRAPH Computer Graphics.* ACM, 1984. p. 119-127.