COSC6372 HW5 – Texture Mapping

Jiechang Guo UHID: 2084258

April 7, 2023

1 Problem

The problem for this assignment is to render the teapot and plane with texture and support both orthogonal projection and perspective projection. The issue of perspective distortion caused by simply linear interpolating between texture coordinates will be addressed.

2 Method

In this assignment, the color for each fragment will be determined by the given texture. For each vertex, a texture coordinate is given as $[u, v]$ and the range of u,v is between $[0, 1]$. During the rasterization process, the texture coordinate for each fragment will be interpolated. Finally, a simple sampler will be used to get the color from the texture.

Orthogonal Projection For orthogonal projection, the interpolating of texture coordinates is as easy as interpolating of vertex positions, linear interpolation will be utilized. The linear interpolating between $[u1, v1]$ and $[u2, v2]$ with $k \in [0, 1]$.

$$u = u1 + (u2 - u1) * k$$
$$v = v1 + (v2 - v1) * k$$

Perspective Projection Unlike the orthogonal projection, the object in the perspective projection will have an effect similar to the real-world scene, which is the further the object is the smaller it looks. In this case, if we interpolate the texture coordinates using linear interpolation will cause observable artifacts. The perspective correctness technique will be used to address this issue. Let $z1, z2$ be the depth component in the view space. The linear interpolation will be done for $u/z, v/z, 1/z$. The perspective correct interpolating between $u1$ and $v1$ with $k \in [0, 1]$ is given as followed, for the interpolation with v is similar[1].

$$u = u1/z1 + (u2/z2 - u1/z1) * k$$
$$z = 1/z1 + (1/z2 - 1/z1) * k$$
$$u\_correct = u * 1/z$$

Simple 2D sampler To sample a pixel color from the given texture, the texture coordinate should be scaled to the texture size. Let *texture* be the 2D array that stores the pixel value for the image. Let *width*, *height* be the size of the image. The pixel color at $[u, v]$ can be calculated as followed.

$$s = u * (width - 1)$$
$$t = v * (height - 1)$$
$$color = texture[s][t]$$

3 Implementation

Texture coordinate interpolation In the texcoordInterpolate function, simple linear interpolation is used for orthogonal projection. On the other hand, for perspective projection, the perspective correction linear interpolation is utilized. Note that the depth component is in view space. For the detail implementation of triangle rasteriation please check the source code.

```
void GzFrameBuffer::texcoordInterpolate(GzReal key1, GzTexCoord &val1, GzReal key2, GzTexCoord &
    val2, GzReal key,GzReal z1,GzReal z2,  GzTexCoord &val)
{
  GzReal k=(key-key1)/(key2-key1);

  if(isPerspective)
  {
    GzReal u_1 = val1[0]/z1;
    GzReal v_1 = val1[1]/z1;

    GzReal u_2 = val2[0]/z2;
    GzReal v_2 = val2[1]/z2;

    GzReal z_1 = 1.0/z1;
    GzReal z_2 = 1.0/z2;

    GzReal u = u_1 + (u_2 - u_1) * k;
    GzReal v = v_1 + (v_2 - v_1) * k;
    GzReal z = z_1 + (z_2 - z_1) * k;

    z = 1.0/z;
    u = u * z;
    v = v * z;
    val[0] = u;
    val[1] = v;
  }
  else
  {
    for (GzInt i=0; i<2; i++)
    {
      val[i]=val1[i]+(val2[i]-val1[i])*k;
    }
  }
}
```

Simple 2D texture sampler From texture coordinate to pixel color.

```
GzColor GzFrameBuffer::sampleTexture(GzTexCoord texCoord)
{
    GzInt width = texture1.sizeW()-1;
  GzInt height = texture1.sizeH()-1;
  GzInt u = texCoord[0]*width;
  GzInt v = texCoord[1]*height;
  //std::cout<<u<< " " << v << std::endl;
  GzColor color = texture1.get(u,v);
    return color;
}
```

4 Result

In this section, the final result of the rendered images will be illustrated.
The final result of texture mapping in orthogonal projection is shown in figure 1, and the final result of texture mapping in perspective projection is shown in figure 2.

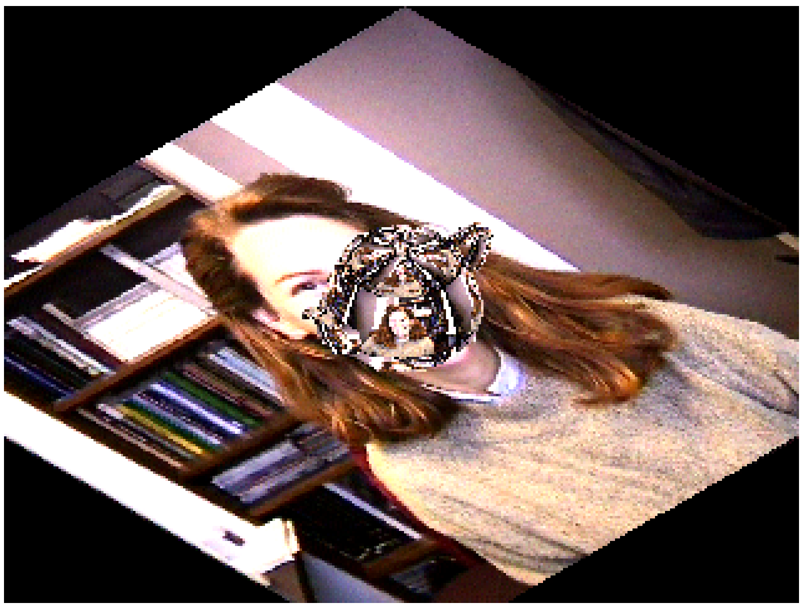

Figure 1: Orthogonal projection

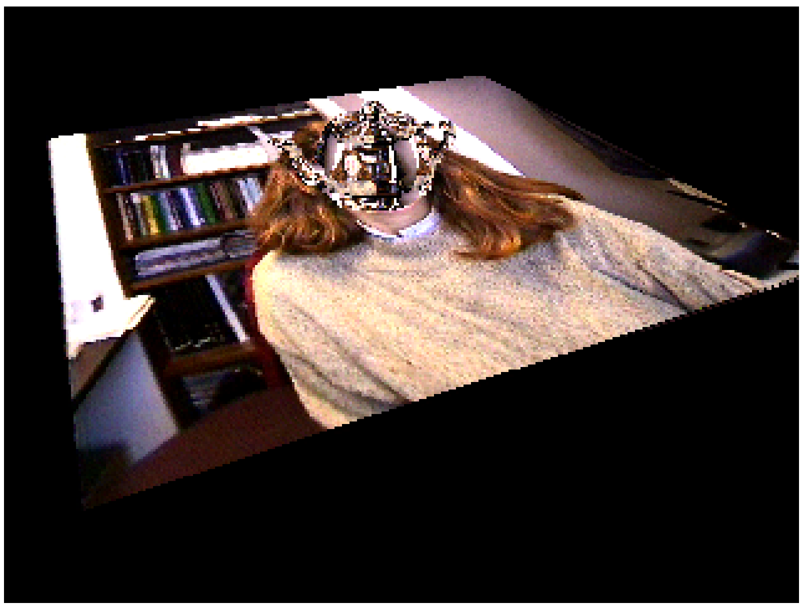

Figure 2: Perspective projection

The result for perspective correction is shown figure 3. The image on the left is using simple linear interpolation on the texture coordinates. The image on the right is using the perspective technique mentioned before. The artifacts on the plane are fixed after perspective correction is applied.

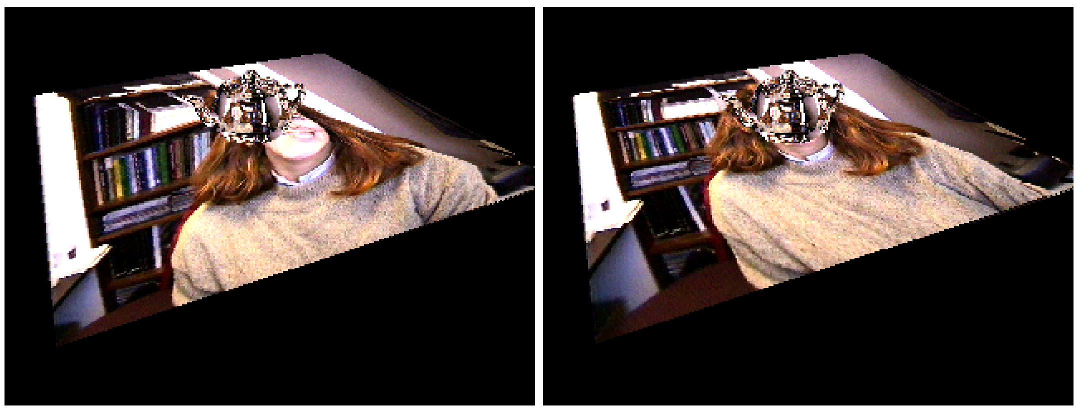

Figure 3: Perspective correctness result. Left: texture mapping with distortion. Right: texture mapping after perspective correction.

References

[1] wikipedia. *Perspective correctness.* `https://en.wikipedia.org/wiki/Texture_mapping#Perspective_correctness`.