Assignment 5: Texture Mapping

Mai Trinh
COSC 6372 Computer Graphics*, University of Houston

April 16, 2021

1 Problem

The goal of this assignment is to add the texture mapping function into the *Gz* library. Given an input file containing the information of several triangles including the coordinates of 3 vertices, and its texture coordinate, and a texture file, the application should be able to apply texture to image for rendering. Several classes, functions, and data types are already given in the *Gz* library. We were to complete the implementation of the functions in *Gz* class and *GzFrameBuffer* class to enable the texture mapping function and produce the expected output images.

2 Methods

The main challenge of this assignment is to implement functions to support simple texture mapping without shading/lighting. The program needs to support both orthogonal projection and perspective projection.

For the orthogonal project, we use a simple affine texture mapping algorithm which linearly interpolates texture coordinates across triangle's vertices [2]. Basic algorithm for affine texture mapping:

```
for each pixel in the rasterized image:
– interpolate coordinates across triangle
– sample (evaluate) texture at interpolated
– set color of fragment to sampled texture value
```

However, affine texture mapping algorithm can create noticeable distortion when used with perspective transformation. Due to perspective projection (homogeneous divide), Barycentric interpolation of values on a triangle with different depth is not an affine function of screen XY coordinates[1]. For perspective projection, we apply perspective correct interpolation to avoid artifact. Basic algorithm for perspective correct interpolation:

```
Goal: interpolate some attribute φ at vertices
Basic algorithm:
– Compute depth z at each vertex
– Evaluate Z := 1/z and P := φ/z at each vertex
– Interpolate Z and P using standard (2D) barycentric coords
– At each fragment, divide interpolated P by interpolated Z to get final value
```

3 Implementation

Assume that the task of reading input from a file and storing it in appropriate vectors are correctly implemented in the main application. The coordinates *(x,y,z)* of each pixel is stored in *vertexQueue*, thetexture coordinate of the vertex is stored in *texCoordQueue* in *Gz* library. When GZ_TEXTURE is enabled, we pass to the framebuffer the texture image as well as vertex information including coordinates and texture coordinates. Note that we only consider texture mapping without lighting, so we need to make sure that GZ_LIGHTING is off. We also only implement the triangle rasterization, so we need to make sure current primitive is GZ_TRIANGLES.

3.1 Affine texture mapping

Affine texture mapping linearly interpolates texture coordinates across a surface, and so is the fastest form of texture mapping [2]. To linearly interpolate texture coordinate, we implement the function based on linearly interpolation between 2 known points method (as described in previous assignment) with minor changes. The implementation for linearly interpolating texture coordinates is shown below.

```
void GzFrameBuffer::textureInterpolate(GzReal key1, GzTexCoord& val1, GzReal key2,
                                       GzTexCoord& val2, GzReal key, GzTexCoord& val) {
    GzReal k = (key - key1) / (key2 - key1);
    for (GzInt i = 0; i < 2; i++)
        val[i] = val1[i] + (val2[i] - val1[i]) * k;}
```

To implement affine texture mapping, I follow the similar outline and code as previous assignment in which the rasterizer linearly interpolate texture coordinates *u,v* accross the triangle as it does with *z* or color as shown in *drawTriangleWTexture* and *drawRasLineWTexture* functions.

*Course assignments

```
void GzFrameBuffer::drawTriangleWTexture(vector<GzVertex>& v, vector<GzTexCoord> t, GzFunctional status) {
    GzInt yMin, yMax;
    GzReal xMin, xMax, zMin, zMax;
    GzTexCoord tMin, tMax;
    v.push_back(v[0]);
    t.push_back(t[0]);
    yMin = INT_MAX;
    yMax = -INT_MAX;
    for (GzInt i = 0; i < 3; i++) {
        yMin = min((GzInt)floor(v[i][Y]), yMin);
        yMax = max((GzInt)floor(v[i][Y] - 1e-3), yMax);
    }
    for (GzInt y = yMin; y <= yMax; y++) {
        xMin = INT_MAX;
        xMax = -INT_MAX;
        for (GzInt i = 0; i < 3; i++) {
            if ((GzInt)floor(v[i][Y]) == y) {
                if (v[i][X] < xMin) {
                    xMin = v[i][X];
                    zMin = v[i][Z];
                    tMin = t[i];
                } if (v[i][X] > xMax) {
                    xMax = v[i][X];
                    zMax = v[i][Z];
                    tMax = t[i];}}
            if ((y - v[i][Y]) * (y - v[i + 1][Y]) < 0) {
                GzReal x;
                realInterpolate(v[i][Y], v[i][X], v[i + 1][Y], v[i + 1][X], y, x);
                if (x < xMin) {
                    xMin = x;
                    realInterpolate(v[i][Y], v[i][Z], v[i + 1][Y], v[i + 1][Z], y, zMin);
                    textureInterpolate(v[i][Y], t[i], v[i + 1][Y], t[i + 1], y, tMin);
                } if (x > xMax) {
                    xMax = x;
                    realInterpolate(v[i][Y], v[i][Z], v[i + 1][Y], v[i + 1][Z], y, zMax);
                    textureInterpolate(v[i][Y], t[i], v[i + 1][Y], t[i + 1], y, tMax);}}}
        drawRasLineWTexture(y, xMin, zMin, tMin, xMax - 1e-3, zMax, tMax, status);
    }
}

// Important note: to know the UV texture coordinate of a pixel, we interpolate texture coordinates
// of the vertices around it and use that UV to look up the color in the texture
void GzFrameBuffer::drawRasLineWTexture(GzInt y, GzReal xMin, GzReal zMin, GzTexCoord& tMin,
                                        GzReal xMax, GzReal zMax, GzTexCoord& tMax,
                                        GzFunctional status) {
    if ((y < 0) || (y >= image.sizeH())) return;
    if ((GzInt)floor(xMin) == (GzInt)floor(xMax)) {
        if (zMin > zMax) drawPoint(GzVertex(floor(xMin), y, zMin), _texture.getTextureColor(tMin), status);
        else drawPoint(GzVertex(floor(xMin), y, zMax), _texture.getTextureColor(tMax), status);
    } else {
        GzReal z;
        GzTexCoord t;
        y = image.sizeH() - y - 1;
        int w = image.sizeW();
        if (status & GZ_DEPTH_TEST) {
            for (int x = max(0, (GzInt)floor(xMin)); x <= min(w - 1, (GzInt)floor(xMax)); x++) {
                realInterpolate(xMin, zMin, xMax, zMax, x, z);
                if (z >= depthBuffer[x][y]) {
                        textureInterpolate(xMin, tMin, xMax, tMax, x, t);
                        image.set(x, y, _texture.getTextureColor(t));
                        depthBuffer[x][y] = z;}}
        } else {
            for (int x = max(0, (GzInt)floor(xMin)); x <= min(w - 1, (GzInt)floor(xMax)); x++) {
                realInterpolate(xMin, zMin, xMax, zMax, x, z);
                textureInterpolate(xMin, tMin, xMax, tMax, x, t);
                image.set(x, y, _texture.getTextureColor(t));
                depthBuffer[x][y] = z;}}}}
```

3.1.1 Lookup texture color

For each fragment, we use the interpolated texture coordinate *u,v* values to look up the texture color as shown in *getTextureColor* function.

```
GzColor GzImage::getTextureColor(GzTexCoord t) {
        if ((t[U] < 0) || (t[V] < 0) || (t[U] > 1) || (t[V] > 1)) return GzColor();
        int w = sizeW();
```

```
int h = sizeH();
//Scale to texture size
int x = (int)(t[U] * (w - 1));
int y = (int)(t[V] * (h - 1));
return pixel[x][y];}
```

3.2 Perspective correct interpolation

To achieve perspective-correct texture mapping, we need to find the texture coordinates in a more complex method. Similar to the affine texture mapping, we first interpolate the texture coordinates, and then do the perspective divide. Rather than interpolating *u,v* directly, interpolate *u/z, v/z*. $1/z$ is also interpolated. So at each vertex, we divide the texture coordinates by z, where z is the depth component of the vertex.

4 Results

The results of the implementation are shown below. Left image displays the result of affine texture mapping, and the right image shows the result of perspective correct texture mapping. As you can see, the teapot is applied with texture provided. The affine texture mapping in the first image is correct and no artifact. In the second image, there appears to be artifact which is the result of perspective distortion. Thus it seems that the perspective correct interpolation were applied incorrectly.

Figure 1: Texture Mapping - Result images

References

[1] *Perspective Projection and Texture Mapping*. URL: http://15462.courses.cs.cmu.edu/fall2020content/lectures/07_texture/07_texture_slides.pdf.

[2] *Texture mapping*. URL: https://en.wikipedia.org/wiki/Texture_mapping#Perspective_correctness.