ssloy / tinyrenderer

Branch: master ▾ tinyrenderer / main.cpp Find file Copy path

ssloy tangent space normal mapping 907bb56 on Mar 13, 2016

2 contributors

95 lines (75 sloc) 3.04 KB

```
#include <vector>
#include <limits>
#include <iostream>
#include "tgaimage.h"
#include "model.h"
#include "geometry.h"
#include "our_gl.h"

Model *model        = NULL;

const int width  = 800;
const int height = 800;

Vec3f light_dir(1,1,1);
Vec3f       eye(1,1,3);
Vec3f    center(0,0,0);
Vec3f        up(0,1,0);

struct Shader : public IShader {
    mat<2,3,float> varying_uv;  // triangle uv coordinates, written by the vertex shader, read by the fragment shader
    mat<4,3,float> varying_tri; // triangle coordinates (clip coordinates), written by VS, read by FS
    mat<3,3,float> varying_nrm; // normal per vertex to be interpolated by FS
    mat<3,3,float> ndc_tri;     // triangle in normalized device coordinates

    virtual Vec4f vertex(int iface, int nthvert) {
        varying_uv.set_col(nthvert, model->uv(iface, nthvert));
        varying_nrm.set_col(nthvert, proj<3>((Projection*ModelView).invert_transpose()*embed<4>(model->normal(iface, nthvert), 
        Vec4f gl_Vertex = Projection*ModelView*embed<4>(model->vert(iface, nthvert));
        varying_tri.set_col(nthvert, gl_Vertex);
        ndc_tri.set_col(nthvert, proj<3>(gl_Vertex/gl_Vertex[3]));
        return gl_Vertex;
    }

    virtual bool fragment(Vec3f bar, TGAColor &color) {
        Vec3f bn = (varying_nrm*bar).normalize();
        Vec2f uv = varying_uv*bar;

        mat<3,3,float> A;
        A[0] = ndc_tri.col(1) - ndc_tri.col(0);
        A[1] = ndc_tri.col(2) - ndc_tri.col(0);
        A[2] = bn;

        mat<3,3,float> AI = A.invert();

        Vec3f i = AI * Vec3f(varying_uv[0][1] - varying_uv[0][0], varying_uv[0][2] - varying_uv[0][0], 0);
        Vec3f j = AI * Vec3f(varying_uv[1][1] - varying_uv[1][0], varying_uv[1][2] - varying_uv[1][0], 0);

        mat<3,3,float> B;
        B.set_col(0, i.normalize());
        B.set_col(1, j.normalize());
        B.set_col(2, bn);

        Vec3f n = (B*model->normal(uv)).normalize();

```

```
        float diff = std::max(0.f, n*light_dir);
        color = model->diffuse(uv)*diff;

        return false;
    }
};

int main(int argc, char** argv) {
    if (2>argc) {
        std::cerr << "Usage: " << argv[0] << " obj/model.obj" << std::endl;
        return 1;
    }

    float *zbuffer = new float[width*height];
    for (int i=width*height; i--; zbuffer[i] = -std::numeric_limits<float>::max());

    TGAImage frame(width, height, TGAImage::RGB);
    lookat(eye, center, up);
    viewport(width/8, height/8, width*3/4, height*3/4);
    projection(-1.f/(eye-center).norm());
    light_dir = proj<3>((Projection*ModelView*embed<4>(light_dir, 0.f))).normalize();

    for (int m=1; m<argc; m++) {
        model = new Model(argv[m]);
        Shader shader;
        for (int i=0; i<model->nfaces(); i++) {
            for (int j=0; j<3; j++) {
                shader.vertex(i, j);
            }
            triangle(shader.varying_tri, shader, frame, zbuffer);
        }
        delete model;
    }
    frame.flip_vertically(); // to place the origin in the bottom left corner of the image
    frame.write_tga_file("framebuffer.tga");

    delete [] zbuffer;
    return 0;
}
```