**TUGAS BESAR GRAFIKA KOMPUTER**

**KOMPUTER APPLE 3D**

**Dosen : Hendri Karisma, S.Kom.**



**Disusun Oleh :**

Irfan Budi Santoso - 10109903

Budi Arisandi - 10109911

Hendra - 10109928

Kelas : IF17K

**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK DAN ILMU KOMPUTER**

**UNIVERSITAS KOMPUTER INDONESIA**

**BANDUNG**

**2013**

**BAB I**

**PENDAHULUAN**

* 1. **LATAR BELAKANG**

Pada saat ini grafika komputer sudah digunakan pada bidang sains, Design, engineering, kedokteran, bisnis, industri, pemerintahan, seni, hiburan, iklan, pendidikan, dan lain-lain. Oleh karena itu penulis mengambil judul “KOMPUTER APPLE 3D”. Dalam rangka mengimplementasikan grafika komputer di dalam salah satu bidang yaitu bidang Design.

Untuk pembuatan Komputer Apple 3D, penulis merancang detail bentuk komputer desktop tersebut dengan beberapa objek yang ada di dalam sepeda itu sendiri yaitu diantaranya :

1. Monitor yang di dalamnya terdapat objek : Layar, body monitor, stand monitor
2. Keyboard yang di dalamnya terdapat objek : papan keyboard, tombol
3. Mouse yang di dalamnya terdapat objek : body mouse, scroll
4. Kabel

Latar belakang penulis mengambil tema Komputer Apple 3D yaitu penulis ingin mencoba mengkombinasikan beberapa bentuk benda yang akan kami buat dalam bentuk 3D sehingga bentuk-bentuk tersebut dapat disajikan ke dalam satu tampilan layar.

* 1. **IDENTIFIKASI MASALAH**

Berdasarkan latar belakang dapat didefinisikan beberapa masalah yaitu :

1. Bagaimana cara memodelkan Komputer Desktop dalam bentuk 3D pada openGL dengan menggunakan IDE Microsoft Visual C++ 6.0 yang mendukung openGL.
2. Bagaimana membuat Komputer Desktop 3D sehingga Objek yang dibuat tersebut dapat dilihat dari berbagai arah.

* 1. **MAKSUD DAN TUJUAN**

Berdasarkan permasalahan yang ada, maka maksud dari penulisan tugas besar ini adalah untuk membangun Komputer Apple 3D dengan berbasis desktop. Sedangkan tujuan yang akan dicapai adalah :

1. Menciptakan Komputer Desktop dalam bentuk 3D.
2. Membuat Komputer Apple 3D .
3. Mengaplikasikan materi openGL dari mata kuliah Grafika Komputer.
   1. **Objek Pembangun**

Objek pembangun yang akan digunakan dalam pemodelan Komputer Apple 3D adalah :

1. GlmMax
2. GlmAbs
3. GlmDot
4. GlmCross
   1. **BATASAN MASALAH**

Dalam pembahasan tugas Komputer Apple 3D dibatasi sebagai berikut :

1. Bahasa yang digunakan bahasa pemrograman C++
2. IDE yang digunakan Visual C++ 6.0 dengan openGL
3. Program ini hanya menampilkan Komputer Apple 3D
4. Komputer yang akan dibangun dapat dilihat dari semua arah.

**BAB II**

**KONTRIBUSI TUGAS**

1. **Source Code Irfan Budi Santoso**

**static** GLfloat

**glmMax**(GLfloat a, GLfloat b)

{

**if** (b **>** a)

**return** b;

**return** a;

}

**static** GLfloat

**glmAbs**(GLfloat f)

{

**if** (f **<** 0)

**return** **-**f;

**return** f;

}

**static** GLfloat

**glmDot**(GLfloat**\*** u, GLfloat**\*** v)

{

    assert(u); assert(v);

**return** u[0]**\***v[0] **+** u[1]**\***v[1] **+** u[2]**\***v[2];

}

**static** GLvoid

**glmCross**(GLfloat**\*** u, GLfloat**\*** v, GLfloat**\*** n)

{

    assert(u); assert(v); assert(n);

    n[0] **=** u[1]**\***v[2] **-** u[2]**\***v[1];

    n[1] **=** u[2]**\***v[0] **-** u[0]**\***v[2];

    n[2] **=** u[0]**\***v[1] **-** u[1]**\***v[0];

}

**static** GLvoid

**glmNormalize**(GLfloat**\*** v)

{

    GLfloat l;

    assert(v);

    l **=** (GLfloat)sqrt(v[0]**\***v[0] **+** v[1]**\***v[1] **+** v[2]**\***v[2]);

    v[0] **/=** l;

    v[1] **/=** l;

    v[2] **/=** l;

}

**static** GLboolean

**glmEqual**(GLfloat**\*** u, GLfloat**\*** v, GLfloat epsilon)

{

**if** (glmAbs(u[0] **-** v[0]) **<** epsilon **&&**

        glmAbs(u[1] **-** v[1]) **<** epsilon **&&**

        glmAbs(u[2] **-** v[2]) **<** epsilon)

    {

**return** GL\_TRUE;

    }

**return** GL\_FALSE;

}

GLfloat**\***

**glmWeldVectors**(GLfloat**\*** vectors, GLuint**\*** numvectors, GLfloat epsilon)

{

    GLfloat**\*** copies;

    GLuint copied;

    GLuint i, j;

    copies **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\*** 3 **\*** (**\***numvectors **+** 1));

    memcpy(copies, vectors, (**sizeof**(GLfloat) **\*** 3 **\*** (**\***numvectors **+** 1)));

    copied **=** 1;

**for** (i **=** 1; i **<=** **\***numvectors; i**++**) {

**for** (j **=** 1; j **<=** copied; j**++**) {

**if** (glmEqual(**&**vectors[3 **\*** i], **&**copies[3 **\*** j], epsilon)) {

**goto** duplicate;

            }

        }

        copies[3 **\*** copied **+** 0] **=** vectors[3 **\*** i **+** 0];

        copies[3 **\*** copied **+** 1] **=** vectors[3 **\*** i **+** 1];

        copies[3 **\*** copied **+** 2] **=** vectors[3 **\*** i **+** 2];

        j **=** copied;

        copied**++**;

duplicate:

        vectors[3 **\*** i **+** 0] **=** (GLfloat)j;

    }

**\***numvectors **=** copied**-**1;

**return** copies;

}

GLMgroup**\***

**glmFindGroup**(GLMmodel**\*** model, **char\*** name)

{

    GLMgroup**\*** group;

    assert(model);

    group **=** model**->**groups;

**while**(group) {

**if** (**!**strcmp(name, group**->**name))

**break**;

        group **=** group**->**next;

    }

**return** group;

}

GLMgroup**\***

**glmAddGroup**(GLMmodel**\*** model, **char\*** name)

{

    GLMgroup**\*** group;

    group **=** glmFindGroup(model, name);

**if** (**!**group) {

        group **=** (GLMgroup**\***)malloc(**sizeof**(GLMgroup));

        group**->**name **=** strdup(name);

        group**->**material **=** 0;

        group**->**numtriangles **=** 0;

        group**->**triangles **=** NULL;

        group**->**next **=** model**->**groups;

        model**->**groups **=** group;

        model**->**numgroups**++**;

    }

**return** group;

}

GLuint

**glmFindMaterial**(GLMmodel**\*** model, **char\*** name)

{

    GLuint i;

**for** (i **=** 0; i **<** model**->**nummaterials; i**++**) {

**if** (**!**strcmp(model**->**materials[i].name, name))

**goto** found;

    }

    printf("glmFindMaterial(): can't find material \"%s\".\n", name);

    i **=** 0;

found:

**return** i;

}

**static** **char\***

**glmDirName**(**char\*** path)

{

**char\*** dir;

**char\*** s;

    dir **=** strdup(path);

    s **=** strrchr(dir, '/');

**if** (s)

        s[1] **=** '\0';

**else**

        dir[0] **=** '\0';

**return** dir;

}

**static** GLvoid

**glmReadMTL**(GLMmodel**\*** model, **char\*** name)

{

**FILE\*** file;

**char\*** dir;

**char\*** filename;

**char** buf[128];

    GLuint nummaterials, i;

    dir **=** glmDirName(model**->**pathname);

    filename **=** (**char\***)malloc(**sizeof**(**char**) **\*** (strlen(dir) **+** strlen(name) **+** 1));

    strcpy(filename, dir);

    strcat(filename, name);

    free(dir);

    file **=** fopen(filename, "r");

**if** (**!**file) {

        fprintf(stderr, "glmReadMTL() failed: can't open material file \"%s\".\n",

            filename);

        exit(1);

    }

    free(filename);

    nummaterials **=** 1;

**while**(fscanf(file, "%s", buf) **!=** EOF) {

**switch**(buf[0]) {

**case** '#':

            fgets(buf, **sizeof**(buf), file);

**break**;

**case** 'n':

            fgets(buf, **sizeof**(buf), file);

            nummaterials**++**;

            sscanf(buf, "%s %s", buf, buf);

**break**;

        default:

            fgets(buf, **sizeof**(buf), file);

**break**;

        }

    }

    rewind(file);

    model**->**materials **=** (GLMmaterial**\***)malloc(**sizeof**(GLMmaterial) **\*** nummaterials);

    model**->**nummaterials **=** nummaterials;

**for** (i **=** 0; i **<** nummaterials; i**++**) {

        model**->**materials[i].name **=** NULL;

        model**->**materials[i].shininess **=** 65.0;

        model**->**materials[i].diffuse[0] **=** 0.8;

        model**->**materials[i].diffuse[1] **=** 0.8;

        model**->**materials[i].diffuse[2] **=** 0.8;

        model**->**materials[i].diffuse[3] **=** 1.0;

        model**->**materials[i].ambient[0] **=** 0.2;

        model**->**materials[i].ambient[1] **=** 0.2;

        model**->**materials[i].ambient[2] **=** 0.2;

        model**->**materials[i].ambient[3] **=** 1.0;

        model**->**materials[i].specular[0] **=** 0.0;

        model**->**materials[i].specular[1] **=** 0.0;

        model**->**materials[i].specular[2] **=** 0.0;

        model**->**materials[i].specular[3] **=** 1.0;

    }

    model**->**materials[0].name **=** strdup("default");

    nummaterials **=** 0;

**while**(fscanf(file, "%s", buf) **!=** EOF) {

**switch**(buf[0]) {

**case** '#':

            fgets(buf, **sizeof**(buf), file);

**break**;

**case** 'n':

            fgets(buf, **sizeof**(buf), file);

            sscanf(buf, "%s %s", buf, buf);

            nummaterials**++**;

            model**->**materials[nummaterials].name **=** strdup(buf);

**break**;

**case** 'N':

            fscanf(file, "%f", **&**model**->**materials[nummaterials].shininess);

            model**->**materials[nummaterials].shininess **/=** 1000.0;

            model**->**materials[nummaterials].shininess **\*=** 128.0;

**break**;

**case** 'K':

**switch**(buf[1]) {

**case** 'd':

                fscanf(file, "%f %f %f",

**&**model**->**materials[nummaterials].diffuse[0],

**&**model**->**materials[nummaterials].diffuse[1],

**&**model**->**materials[nummaterials].diffuse[2]);

**break**;

**case** 's':

                fscanf(file, "%f %f %f",

**&**model**->**materials[nummaterials].specular[0],

**&**model**->**materials[nummaterials].specular[1],

**&**model**->**materials[nummaterials].specular[2]);

**break**;

**case** 'a':

                fscanf(file, "%f %f %f",

**&**model**->**materials[nummaterials].ambient[0],

**&**model**->**materials[nummaterials].ambient[1],

**&**model**->**materials[nummaterials].ambient[2]);

**break**;

            default:

                fgets(buf, **sizeof**(buf), file);

**break**;

            }

**break**;

            default:

                fgets(buf, **sizeof**(buf), file);

**break**;

        }

    }

}

**static** GLvoid

**glmWriteMTL**(GLMmodel**\*** model, **char\*** modelpath, **char\*** mtllibname)

{

**FILE\*** file;

**char\*** dir;

**char\*** filename;

    GLMmaterial**\*** material;

    GLuint i;

    dir **=** glmDirName(modelpath);

    filename **=** (**char\***)malloc(**sizeof**(**char**) **\*** (strlen(dir)**+**strlen(mtllibname)));

    strcpy(filename, dir);

    strcat(filename, mtllibname);

    free(dir);

    file **=** fopen(filename, "w");

**if** (**!**file) {

        fprintf(stderr, "glmWriteMTL() failed: can't open file \"%s\".\n",

            filename);

        exit(1);

    }

    free(filename);

    fprintf(file, "# \n");

    fprintf(file, "# Wavefront MTL generated by GLM library\n");

    fprintf(file, "# \n");

    fprintf(file, "# GLM library\n");

    fprintf(file, "# Nate Robins\n");

    fprintf(file, "# ndr@pobox.com\n");

    fprintf(file, "# http://www.pobox.com/~ndr\n");

    fprintf(file, "# \n\n");

**for** (i **=** 0; i **<** model**->**nummaterials; i**++**) {

        material **=** **&**model**->**materials[i];

        fprintf(file, "newmtl %s\n", material**->**name);

        fprintf(file, "Ka %f %f %f\n",

            material**->**ambient[0], material**->**ambient[1], material**->**ambient[2]);

        fprintf(file, "Kd %f %f %f\n",

            material**->**diffuse[0], material**->**diffuse[1], material**->**diffuse[2]);

        fprintf(file, "Ks %f %f %f\n",

            material**->**specular[0],material**->**specular[1],material**->**specular[2]);

        fprintf(file, "Ns %f\n", material**->**shininess **/** 128.0 **\*** 1000.0);

        fprintf(file, "\n");

    }

}

**static** GLvoid

**glmFirstPass**(GLMmodel**\*** model, **FILE\*** file)

{

    GLuint numvertices;

    GLuint numnormals;

    GLuint numtexcoords;

    GLuint numtriangles;

    GLMgroup**\*** group;

**int** v, n, t;

**char** buf[128];

    group **=** glmAddGroup(model, "default");

    numvertices **=** numnormals **=** numtexcoords **=** numtriangles **=** 0;

**while**(fscanf(file, "%s", buf) **!=** EOF) {

**switch**(buf[0]) {

**case** '#':

            fgets(buf, **sizeof**(buf), file);

**break**;

**case** 'v':

**switch**(buf[1]) {

**case** '\0':

                fgets(buf, **sizeof**(buf), file);

                numvertices**++**;

**break**;

**case** 'n':

                fgets(buf, **sizeof**(buf), file);

                numnormals**++**;

**break**;

**case** 't':

                fgets(buf, **sizeof**(buf), file);

                numtexcoords**++**;

**break**;

            default:

                printf("glmFirstPass(): Unknown token \"%s\".\n", buf);

                exit(1);

**break**;

            }

**break**;

**case** 'm':

                fgets(buf, **sizeof**(buf), file);

                sscanf(buf, "%s %s", buf, buf);

                model**->**mtllibname **=** strdup(buf);

                glmReadMTL(model, buf);

**break**;

**case** 'u':

                fgets(buf, **sizeof**(buf), file);

**break**;

**case** 'g':

                fgets(buf, **sizeof**(buf), file);

**#if SINGLE\_STRING\_GROUP\_NAMES**

                sscanf(buf, "%s", buf);

**#else**

                buf[strlen(buf)**-**1] **=** '\0';

**#endif**

                group **=** glmAddGroup(model, buf);

**break**;

**case** 'f':

                v **=** n **=** t **=** 0;

                fscanf(file, "%s", buf);

**if** (strstr(buf, "//")) {

                    sscanf(buf, "%d//%d", **&**v, **&**n);

                    fscanf(file, "%d//%d", **&**v, **&**n);

                    fscanf(file, "%d//%d", **&**v, **&**n);

                    numtriangles**++**;

                    group**->**numtriangles**++**;

**while**(fscanf(file, "%d//%d", **&**v, **&**n) **>** 0) {

                        numtriangles**++**;

                        group**->**numtriangles**++**;

                    }

                } **else** **if** (sscanf(buf, "%d/%d/%d", **&**v, **&**t, **&**n) **==** 3) {

                    fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);

                    fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);

                    numtriangles**++**;

                    group**->**numtriangles**++**;

**while**(fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n) **>** 0) {

                        numtriangles**++**;

                        group**->**numtriangles**++**;

                    }

                } **else** **if** (sscanf(buf, "%d/%d", **&**v, **&**t) **==** 2) {

                    fscanf(file, "%d/%d", **&**v, **&**t);

                    fscanf(file, "%d/%d", **&**v, **&**t);

                    numtriangles**++**;

                    group**->**numtriangles**++**;

**while**(fscanf(file, "%d/%d", **&**v, **&**t) **>** 0) {

                        numtriangles**++**;

                        group**->**numtriangles**++**;

                    }

                } **else** {

                    fscanf(file, "%d", **&**v);

                    fscanf(file, "%d", **&**v);

                    numtriangles**++**;

                    group**->**numtriangles**++**;

**while**(fscanf(file, "%d", **&**v) **>** 0) {

                        numtriangles**++**;

                        group**->**numtriangles**++**;

                    }

                }

**break**;

            default:

                fgets(buf, **sizeof**(buf), file);

**break**;

        }

  }

  model**->**numvertices **=** numvertices;

  model**->**numnormals **=** numnormals;

  model**->**numtexcoords **=** numtexcoords;

  model**->**numtriangles **=** numtriangles;

  group **=** model**->**groups;

**while**(group) {

      group**->**triangles **=** (GLuint**\***)malloc(**sizeof**(GLuint) **\*** group**->**numtriangles);

      group**->**numtriangles **=** 0;

      group **=** group**->**next;

  }

}

**static** GLvoid

**glmSecondPass**(GLMmodel**\*** model, **FILE\*** file)

{

    GLuint numvertices;

    GLuint numnormals;

    GLuint numtexcoords;

    GLuint numtriangles;

    GLfloat**\*** vertices;

    GLfloat**\*** normals;

    GLfloat**\*** texcoords;

    GLMgroup**\*** group;

    GLuint material;

**int** v, n, t;

**char** buf[128];

    vertices **=** model**->**vertices;

    normals **=** model**->**normals;

    texcoords **=** model**->**texcoords;

    group **=** model**->**groups;

    numvertices **=** numnormals **=** numtexcoords **=** 1;

    numtriangles **=** 0;

    material **=** 0;

**while**(fscanf(file, "%s", buf) **!=** EOF) {

**switch**(buf[0]) {

**case** '#':

            fgets(buf, **sizeof**(buf), file);

**break**;

**case** 'v':

**switch**(buf[1]) {

**case** '\0':

                fscanf(file, "%f %f %f",

**&**vertices[3 **\*** numvertices **+** 0],

**&**vertices[3 **\*** numvertices **+** 1],

**&**vertices[3 **\*** numvertices **+** 2]);

                numvertices**++**;

**break**;

**case** 'n':

                fscanf(file, "%f %f %f",

**&**normals[3 **\*** numnormals **+** 0],

**&**normals[3 **\*** numnormals **+** 1],

**&**normals[3 **\*** numnormals **+** 2]);

                numnormals**++**;

**break**;

**case** 't':

                fscanf(file, "%f %f",

**&**texcoords[2 **\*** numtexcoords **+** 0],

**&**texcoords[2 **\*** numtexcoords **+** 1]);

                numtexcoords**++**;

**break**;

            }

**break**;

**case** 'u':

                fgets(buf, **sizeof**(buf), file);

                sscanf(buf, "%s %s", buf, buf);

                group**->**material **=** material **=** glmFindMaterial(model, buf);

**break**;

**case** 'g':

                fgets(buf, **sizeof**(buf), file);

**#if SINGLE\_STRING\_GROUP\_NAMES**

                sscanf(buf, "%s", buf);

**#else**

                buf[strlen(buf)**-**1] **=** '\0';

**#endif**

                group **=** glmFindGroup(model, buf);

                group**->**material **=** material;

**break**;

**case** 'f':

                v **=** n **=** t **=** 0;

                fscanf(file, "%s", buf);

**if** (strstr(buf, "//")) {

                    sscanf(buf, "%d//%d", **&**v, **&**n);

                    T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).nindices[0] **=** n **<** 0 **?** n **+** numnormals **:** n;

                    fscanf(file, "%d//%d", **&**v, **&**n);

                    T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).nindices[1] **=** n **<** 0 **?** n **+** numnormals **:** n;

                    fscanf(file, "%d//%d", **&**v, **&**n);

                    T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;

                    group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                    numtriangles**++**;

**while**(fscanf(file, "%d//%d", **&**v, **&**n) **>** 0) {

                        T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];

                        T(numtriangles).nindices[0] **=** T(numtriangles**-**1).nindices[0];

                        T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];

                        T(numtriangles).nindices[1] **=** T(numtriangles**-**1).nindices[2];

                        T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                        T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;

                        group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                        numtriangles**++**;

                    }

                } **else** **if** (sscanf(buf, "%d/%d/%d", **&**v, **&**t, **&**n) **==** 3) {

                    T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).tindices[0] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                    T(numtriangles).nindices[0] **=** n **<** 0 **?** n **+** numnormals **:** n;

                    fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);

                    T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).tindices[1] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                    T(numtriangles).nindices[1] **=** n **<** 0 **?** n **+** numnormals **:** n;

                    fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);

                    T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                    T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;

                    group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                    numtriangles**++**;

**while**(fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n) **>** 0) {

                        T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];

                        T(numtriangles).tindices[0] **=** T(numtriangles**-**1).tindices[0];

                        T(numtriangles).nindices[0] **=** T(numtriangles**-**1).nindices[0];

                        T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];

                        T(numtriangles).tindices[1] **=** T(numtriangles**-**1).tindices[2];

                        T(numtriangles).nindices[1] **=** T(numtriangles**-**1).nindices[2];

                        T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                        T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                        T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;

                        group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                        numtriangles**++**;

                    }

                } **else** **if** (sscanf(buf, "%d/%d", **&**v, **&**t) **==** 2) {

                    T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).tindices[0] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                    fscanf(file, "%d/%d", **&**v, **&**t);

                    T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).tindices[1] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                    fscanf(file, "%d/%d", **&**v, **&**t);

                    T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                    group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                    numtriangles**++**;

**while**(fscanf(file, "%d/%d", **&**v, **&**t) **>** 0) {

                        T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];

                        T(numtriangles).tindices[0] **=** T(numtriangles**-**1).tindices[0];

                        T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];

                        T(numtriangles).tindices[1] **=** T(numtriangles**-**1).tindices[2];

                        T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                        T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;

                        group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                        numtriangles**++**;

                    }

                } **else** {

                    sscanf(buf, "%d", **&**v);

                    T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    fscanf(file, "%d", **&**v);

                    T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    fscanf(file, "%d", **&**v);

                    T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                    group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                    numtriangles**++**;

**while**(fscanf(file, "%d", **&**v) **>** 0) {

                        T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];

                        T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];

                        T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;

                        group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;

                        numtriangles**++**;

                    }

                }

**break**;

            default:

                fgets(buf, **sizeof**(buf), file);

**break**;

    }

  }

**#if 0**

*printf(" Memory: %d bytes\n",*

*numvertices \* 3\*sizeof(GLfloat) +*

*numnormals \* 3\*sizeof(GLfloat) \* (numnormals ? 1 : 0) +*

*numtexcoords \* 3\*sizeof(GLfloat) \* (numtexcoords ? 1 : 0) +*

*numtriangles \* sizeof(GLMtriangle));*

**#endif**

}

1. **Source Code Hendra**

GLfloat

**glmUnitize**(GLMmodel**\*** model)

{

    GLuint i;

    GLfloat maxx, minx, maxy, miny, maxz, minz;

    GLfloat cx, cy, cz, w, h, d;

    GLfloat scale;

    assert(model);

    assert(model**->**vertices);

    maxx **=** minx **=** model**->**vertices[3 **+** 0];

    maxy **=** miny **=** model**->**vertices[3 **+** 1];

    maxz **=** minz **=** model**->**vertices[3 **+** 2];

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

**if** (maxx **<** model**->**vertices[3 **\*** i **+** 0])

            maxx **=** model**->**vertices[3 **\*** i **+** 0];

**if** (minx **>** model**->**vertices[3 **\*** i **+** 0])

            minx **=** model**->**vertices[3 **\*** i **+** 0];

**if** (maxy **<** model**->**vertices[3 **\*** i **+** 1])

            maxy **=** model**->**vertices[3 **\*** i **+** 1];

**if** (miny **>** model**->**vertices[3 **\*** i **+** 1])

            miny **=** model**->**vertices[3 **\*** i **+** 1];

**if** (maxz **<** model**->**vertices[3 **\*** i **+** 2])

            maxz **=** model**->**vertices[3 **\*** i **+** 2];

**if** (minz **>** model**->**vertices[3 **\*** i **+** 2])

            minz **=** model**->**vertices[3 **\*** i **+** 2];

    }

    w **=** glmAbs(maxx) **+** glmAbs(minx);

    h **=** glmAbs(maxy) **+** glmAbs(miny);

    d **=** glmAbs(maxz) **+** glmAbs(minz);

    cx **=** (maxx **+** minx) **/** 2.0;

    cy **=** (maxy **+** miny) **/** 2.0;

    cz **=** (maxz **+** minz) **/** 2.0;

    scale **=** 2.0 **/** glmMax(glmMax(w, h), d);

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

        model**->**vertices[3 **\*** i **+** 0] **-=** cx;

        model**->**vertices[3 **\*** i **+** 1] **-=** cy;

        model**->**vertices[3 **\*** i **+** 2] **-=** cz;

        model**->**vertices[3 **\*** i **+** 0] **\*=** scale;

        model**->**vertices[3 **\*** i **+** 1] **\*=** scale;

        model**->**vertices[3 **\*** i **+** 2] **\*=** scale;

    }

**return** scale;

}

GLvoid

**glmDimensions**(GLMmodel**\*** model, GLfloat**\*** dimensions)

{

    GLuint i;

    GLfloat maxx, minx, maxy, miny, maxz, minz;

    assert(model);

    assert(model**->**vertices);

    assert(dimensions);

    maxx **=** minx **=** model**->**vertices[3 **+** 0];

    maxy **=** miny **=** model**->**vertices[3 **+** 1];

    maxz **=** minz **=** model**->**vertices[3 **+** 2];

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

**if** (maxx **<** model**->**vertices[3 **\*** i **+** 0])

            maxx **=** model**->**vertices[3 **\*** i **+** 0];

**if** (minx **>** model**->**vertices[3 **\*** i **+** 0])

            minx **=** model**->**vertices[3 **\*** i **+** 0];

**if** (maxy **<** model**->**vertices[3 **\*** i **+** 1])

            maxy **=** model**->**vertices[3 **\*** i **+** 1];

**if** (miny **>** model**->**vertices[3 **\*** i **+** 1])

            miny **=** model**->**vertices[3 **\*** i **+** 1];

**if** (maxz **<** model**->**vertices[3 **\*** i **+** 2])

            maxz **=** model**->**vertices[3 **\*** i **+** 2];

**if** (minz **>** model**->**vertices[3 **\*** i **+** 2])

            minz **=** model**->**vertices[3 **\*** i **+** 2];

    }

    dimensions[0] **=** glmAbs(maxx) **+** glmAbs(minx);

    dimensions[1] **=** glmAbs(maxy) **+** glmAbs(miny);

    dimensions[2] **=** glmAbs(maxz) **+** glmAbs(minz);

}

GLvoid

**glmScale**(GLMmodel**\*** model, GLfloat scale)

{

    GLuint i;

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

        model**->**vertices[3 **\*** i **+** 0] **\*=** scale;

        model**->**vertices[3 **\*** i **+** 1] **\*=** scale;

        model**->**vertices[3 **\*** i **+** 2] **\*=** scale;

    }

}

GLvoid

**glmReverseWinding**(GLMmodel**\*** model)

{

    GLuint i, swap;

    assert(model);

**for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {

        swap **=** T(i).vindices[0];

        T(i).vindices[0] **=** T(i).vindices[2];

        T(i).vindices[2] **=** swap;

**if** (model**->**numnormals) {

            swap **=** T(i).nindices[0];

            T(i).nindices[0] **=** T(i).nindices[2];

            T(i).nindices[2] **=** swap;

        }

**if** (model**->**numtexcoords) {

            swap **=** T(i).tindices[0];

            T(i).tindices[0] **=** T(i).tindices[2];

            T(i).tindices[2] **=** swap;

        }

    }

**for** (i **=** 1; i **<=** model**->**numfacetnorms; i**++**) {

        model**->**facetnorms[3 **\*** i **+** 0] **=** **-**model**->**facetnorms[3 **\*** i **+** 0];

        model**->**facetnorms[3 **\*** i **+** 1] **=** **-**model**->**facetnorms[3 **\*** i **+** 1];

        model**->**facetnorms[3 **\*** i **+** 2] **=** **-**model**->**facetnorms[3 **\*** i **+** 2];

    }

**for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {

        model**->**normals[3 **\*** i **+** 0] **=** **-**model**->**normals[3 **\*** i **+** 0];

        model**->**normals[3 **\*** i **+** 1] **=** **-**model**->**normals[3 **\*** i **+** 1];

        model**->**normals[3 **\*** i **+** 2] **=** **-**model**->**normals[3 **\*** i **+** 2];

    }

}

GLvoid

**glmFacetNormals**(GLMmodel**\*** model)

{

    GLuint i;

    GLfloat u[3];

    GLfloat v[3];

    assert(model);

    assert(model**->**vertices);

**if** (model**->**facetnorms)

        free(model**->**facetnorms);

    model**->**numfacetnorms **=** model**->**numtriangles;

    model**->**facetnorms **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***

                       3 **\*** (model**->**numfacetnorms **+** 1));

**for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {

        model**->**triangles[i].findex **=** i**+**1;

        u[0] **=** model**->**vertices[3 **\*** T(i).vindices[1] **+** 0] **-**

            model**->**vertices[3 **\*** T(i).vindices[0] **+** 0];

        u[1] **=** model**->**vertices[3 **\*** T(i).vindices[1] **+** 1] **-**

            model**->**vertices[3 **\*** T(i).vindices[0] **+** 1];

        u[2] **=** model**->**vertices[3 **\*** T(i).vindices[1] **+** 2] **-**

            model**->**vertices[3 **\*** T(i).vindices[0] **+** 2];

        v[0] **=** model**->**vertices[3 **\*** T(i).vindices[2] **+** 0] **-**

            model**->**vertices[3 **\*** T(i).vindices[0] **+** 0];

        v[1] **=** model**->**vertices[3 **\*** T(i).vindices[2] **+** 1] **-**

            model**->**vertices[3 **\*** T(i).vindices[0] **+** 1];

        v[2] **=** model**->**vertices[3 **\*** T(i).vindices[2] **+** 2] **-**

            model**->**vertices[3 **\*** T(i).vindices[0] **+** 2];

        glmCross(u, v, **&**model**->**facetnorms[3 **\*** (i**+**1)]);

        glmNormalize(**&**model**->**facetnorms[3 **\*** (i**+**1)]);

    }

}

GLvoid

**glmVertexNormals**(GLMmodel**\*** model, GLfloat angle)

{

    GLMnode**\*** node;

    GLMnode**\*** tail;

    GLMnode**\*\*** members;

    GLfloat**\*** normals;

    GLuint numnormals;

    GLfloat average[3];

    GLfloat dot, cos\_angle;

    GLuint i, avg;

    assert(model);

    assert(model**->**facetnorms);

    cos\_angle **=** cos(angle **\*** M\_PI **/** 180.0);

**if** (model**->**normals)

        free(model**->**normals);

    model**->**numnormals **=** model**->**numtriangles **\*** 3;

    model**->**normals **=** (GLfloat**\***)malloc(**sizeof**(GLfloat)**\*** 3**\*** (model**->**numnormals**+**1));

    members **=** (GLMnode**\*\***)malloc(**sizeof**(GLMnode**\***) **\*** (model**->**numvertices **+** 1));

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**)

        members[i] **=** NULL;

**for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {

        node **=** (GLMnode**\***)malloc(**sizeof**(GLMnode));

        node**->**index **=** i;

        node**->**next **=** members[T(i).vindices[0]];

        members[T(i).vindices[0]] **=** node;

        node **=** (GLMnode**\***)malloc(**sizeof**(GLMnode));

        node**->**index **=** i;

        node**->**next **=** members[T(i).vindices[1]];

        members[T(i).vindices[1]] **=** node;

        node **=** (GLMnode**\***)malloc(**sizeof**(GLMnode));

        node**->**index **=** i;

        node**->**next **=** members[T(i).vindices[2]];

        members[T(i).vindices[2]] **=** node;

    }

    numnormals **=** 1;

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

        node **=** members[i];

**if** (**!**node)

            fprintf(stderr, "glmVertexNormals(): vertex w/o a triangle\n");

        average[0] **=** 0.0; average[1] **=** 0.0; average[2] **=** 0.0;

        avg **=** 0;

**while** (node) {

            dot **=** glmDot(**&**model**->**facetnorms[3 **\*** T(node**->**index).findex],

**&**model**->**facetnorms[3 **\*** T(members[i]**->**index).findex]);

**if** (dot **>** cos\_angle) {

                node**->**averaged **=** GL\_TRUE;

                average[0] **+=** model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 0];

                average[1] **+=** model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 1];

                average[2] **+=** model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 2];

                avg **=** 1; */\* we averaged at least one normal! \*/*

            } **else** {

                node**->**averaged **=** GL\_FALSE;

            }

            node **=** node**->**next;

        }

**if** (avg) {

            glmNormalize(average);

            model**->**normals[3 **\*** numnormals **+** 0] **=** average[0];

            model**->**normals[3 **\*** numnormals **+** 1] **=** average[1];

            model**->**normals[3 **\*** numnormals **+** 2] **=** average[2];

            avg **=** numnormals;

            numnormals**++**;

        }

        node **=** members[i];

**while** (node) {

**if** (node**->**averaged) {

**if** (T(node**->**index).vindices[0] **==** i)

                    T(node**->**index).nindices[0] **=** avg;

**else** **if** (T(node**->**index).vindices[1] **==** i)

                    T(node**->**index).nindices[1] **=** avg;

**else** **if** (T(node**->**index).vindices[2] **==** i)

                    T(node**->**index).nindices[2] **=** avg;

            } **else** {

                model**->**normals[3 **\*** numnormals **+** 0] **=**

                    model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 0];

                model**->**normals[3 **\*** numnormals **+** 1] **=**

                    model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 1];

                model**->**normals[3 **\*** numnormals **+** 2] **=**

                    model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 2];

**if** (T(node**->**index).vindices[0] **==** i)

                    T(node**->**index).nindices[0] **=** numnormals;

**else** **if** (T(node**->**index).vindices[1] **==** i)

                    T(node**->**index).nindices[1] **=** numnormals;

**else** **if** (T(node**->**index).vindices[2] **==** i)

                    T(node**->**index).nindices[2] **=** numnormals;

                numnormals**++**;

            }

            node **=** node**->**next;

        }

    }

    model**->**numnormals **=** numnormals **-** 1;

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

        node **=** members[i];

**while** (node) {

            tail **=** node;

            node **=** node**->**next;

            free(tail);

        }

    }

    free(members);

    normals **=** model**->**normals;

    model**->**normals **=** (GLfloat**\***)malloc(**sizeof**(GLfloat)**\*** 3**\*** (model**->**numnormals**+**1));

**for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {

        model**->**normals[3 **\*** i **+** 0] **=** normals[3 **\*** i **+** 0];

        model**->**normals[3 **\*** i **+** 1] **=** normals[3 **\*** i **+** 1];

        model**->**normals[3 **\*** i **+** 2] **=** normals[3 **\*** i **+** 2];

    }

    free(normals);

}

GLvoid

**glmLinearTexture**(GLMmodel**\*** model)

{

    GLMgroup **\***group;

    GLfloat dimensions[3];

    GLfloat x, y, scalefactor;

    GLuint i;

    assert(model);

**if** (model**->**texcoords)

        free(model**->**texcoords);

    model**->**numtexcoords **=** model**->**numvertices;

    model**->**texcoords**=**(GLfloat**\***)malloc(**sizeof**(GLfloat)**\***2**\***(model**->**numtexcoords**+**1));

    glmDimensions(model, dimensions);

    scalefactor **=** 2.0 **/**

        glmAbs(glmMax(glmMax(dimensions[0], dimensions[1]), dimensions[2]));

**for**(i **=** 1; i **<=** model**->**numvertices; i**++**) {

        x **=** model**->**vertices[3 **\*** i **+** 0] **\*** scalefactor;

        y **=** model**->**vertices[3 **\*** i **+** 2] **\*** scalefactor;

        model**->**texcoords[2 **\*** i **+** 0] **=** (x **+** 1.0) **/** 2.0;

        model**->**texcoords[2 **\*** i **+** 1] **=** (y **+** 1.0) **/** 2.0;

    }

    group **=** model**->**groups;

**while**(group) {

**for**(i **=** 0; i **<** group**->**numtriangles; i**++**) {

            T(group**->**triangles[i]).tindices[0] **=** T(group**->**triangles[i]).vindices[0];

            T(group**->**triangles[i]).tindices[1] **=** T(group**->**triangles[i]).vindices[1];

            T(group**->**triangles[i]).tindices[2] **=** T(group**->**triangles[i]).vindices[2];

        }

        group **=** group**->**next;

    }

**#if 0**

*printf("glmLinearTexture(): generated %d linear texture coordinates\n",*

*model->numtexcoords);*

**#endif**

}

GLvoid

**glmSpheremapTexture**(GLMmodel**\*** model)

{

    GLMgroup**\*** group;

    GLfloat theta, phi, rho, x, y, z, r;

    GLuint i;

    assert(model);

    assert(model**->**normals);

**if** (model**->**texcoords)

        free(model**->**texcoords);

    model**->**numtexcoords **=** model**->**numnormals;

    model**->**texcoords**=**(GLfloat**\***)malloc(**sizeof**(GLfloat)**\***2**\***(model**->**numtexcoords**+**1));

**for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {

        z **=** model**->**normals[3 **\*** i **+** 0]; */\* re-arrange for pole distortion \*/*

        y **=** model**->**normals[3 **\*** i **+** 1];

        x **=** model**->**normals[3 **\*** i **+** 2];

        r **=** sqrt((x **\*** x) **+** (y **\*** y));

        rho **=** sqrt((r **\*** r) **+** (z **\*** z));

**if**(r **==** 0.0) {

            theta **=** 0.0;

            phi **=** 0.0;

        } **else** {

**if**(z **==** 0.0)

                phi **=** 3.14159265 **/** 2.0;

**else**

                phi **=** acos(z **/** rho);

**if**(y **==** 0.0)

                theta **=** 3.141592365 **/** 2.0;

**else**

                theta **=** asin(y **/** r) **+** (3.14159265 **/** 2.0);

        }

        model**->**texcoords[2 **\*** i **+** 0] **=** theta **/** 3.14159265;

        model**->**texcoords[2 **\*** i **+** 1] **=** phi **/** 3.14159265;

    }

    group **=** model**->**groups;

**while**(group) {

**for** (i **=** 0; i **<** group**->**numtriangles; i**++**) {

            T(group**->**triangles[i]).tindices[0] **=** T(group**->**triangles[i]).nindices[0];

            T(group**->**triangles[i]).tindices[1] **=** T(group**->**triangles[i]).nindices[1];

            T(group**->**triangles[i]).tindices[2] **=** T(group**->**triangles[i]).nindices[2];

        }

        group **=** group**->**next;

    }

}

1. **Source Code Budi Arisandi**

GLvoid

**glmDelete**(GLMmodel**\*** model)

{

    GLMgroup**\*** group;

    GLuint i;

    assert(model);

**if** (model**->**pathname) free(model**->**pathname);

**if** (model**->**mtllibname) free(model**->**mtllibname);

**if** (model**->**vertices) free(model**->**vertices);

**if** (model**->**normals) free(model**->**normals);

**if** (model**->**texcoords) free(model**->**texcoords);

**if** (model**->**facetnorms) free(model**->**facetnorms);

**if** (model**->**triangles) free(model**->**triangles);

**if** (model**->**materials) {

**for** (i **=** 0; i **<** model**->**nummaterials; i**++**)

            free(model**->**materials[i].name);

    }

    free(model**->**materials);

**while**(model**->**groups) {

        group **=** model**->**groups;

        model**->**groups **=** model**->**groups**->**next;

        free(group**->**name);

        free(group**->**triangles);

        free(group);

    }

    free(model);

}

GLMmodel**\***

**glmReadOBJ**(**char\*** filename)

{

    GLMmodel**\*** model;

**FILE\*** file;

    file **=** fopen(filename, "r");

**if** (**!**file) {

        fprintf(stderr, "glmReadOBJ() failed: can't open data file \"%s\".\n",

            filename);

        exit(1);

    }

    model **=** (GLMmodel**\***)malloc(**sizeof**(GLMmodel));

    model**->**pathname **=** strdup(filename);

    model**->**mtllibname **=** NULL;

    model**->**numvertices **=** 0;

    model**->**vertices **=** NULL;

    model**->**numnormals **=** 0;

    model**->**normals **=** NULL;

    model**->**numtexcoords **=** 0;

    model**->**texcoords **=** NULL;

    model**->**numfacetnorms **=** 0;

    model**->**facetnorms **=** NULL;

    model**->**numtriangles **=** 0;

    model**->**triangles **=** NULL;

    model**->**nummaterials **=** 0;

    model**->**materials **=** NULL;

    model**->**numgroups **=** 0;

    model**->**groups **=** NULL;

    model**->**position[0] **=** 0.0;

    model**->**position[1] **=** 0.0;

    model**->**position[2] **=** 0.0;

    glmFirstPass(model, file);

    model**->**vertices **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***

        3 **\*** (model**->**numvertices **+** 1));

    model**->**triangles **=** (GLMtriangle**\***)malloc(**sizeof**(GLMtriangle) **\***

        model**->**numtriangles);

**if** (model**->**numnormals) {

        model**->**normals **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***

            3 **\*** (model**->**numnormals **+** 1));

    }

**if** (model**->**numtexcoords) {

        model**->**texcoords **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***

            2 **\*** (model**->**numtexcoords **+** 1));

    }

    rewind(file);

    glmSecondPass(model, file);

    fclose(file);

**return** model;

}

GLvoid

**glmWriteOBJ**(GLMmodel**\*** model, **char\*** filename, GLuint mode)

{

    GLuint i;

**FILE\*** file;

    GLMgroup**\*** group;

    assert(model);

**if** (mode **&** GLM\_FLAT **&&** **!**model**->**facetnorms) {

        printf("glmWriteOBJ() warning: flat normal output requested "

            "with no facet normals defined.\n");

        mode **&=** **~**GLM\_FLAT;

    }

**if** (mode **&** GLM\_SMOOTH **&&** **!**model**->**normals) {

        printf("glmWriteOBJ() warning: smooth normal output requested "

            "with no normals defined.\n");

        mode **&=** **~**GLM\_SMOOTH;

    }

**if** (mode **&** GLM\_TEXTURE **&&** **!**model**->**texcoords) {

        printf("glmWriteOBJ() warning: texture coordinate output requested "

            "with no texture coordinates defined.\n");

        mode **&=** **~**GLM\_TEXTURE;

    }

**if** (mode **&** GLM\_FLAT **&&** mode **&** GLM\_SMOOTH) {

        printf("glmWriteOBJ() warning: flat normal output requested "

            "and smooth normal output requested (using smooth).\n");

        mode **&=** **~**GLM\_FLAT;

    }

**if** (mode **&** GLM\_COLOR **&&** **!**model**->**materials) {

        printf("glmWriteOBJ() warning: color output requested "

            "with no colors (materials) defined.\n");

        mode **&=** **~**GLM\_COLOR;

    }

**if** (mode **&** GLM\_MATERIAL **&&** **!**model**->**materials) {

        printf("glmWriteOBJ() warning: material output requested "

            "with no materials defined.\n");

        mode **&=** **~**GLM\_MATERIAL;

    }

**if** (mode **&** GLM\_COLOR **&&** mode **&** GLM\_MATERIAL) {

        printf("glmWriteOBJ() warning: color and material output requested "

            "outputting only materials.\n");

        mode **&=** **~**GLM\_COLOR;

    }

    file **=** fopen(filename, "w");

**if** (**!**file) {

        fprintf(stderr, "glmWriteOBJ() failed: can't open file \"%s\" to write.\n",

            filename);

        exit(1);

    }

    fprintf(file, "# \n");

    fprintf(file, "# Wavefront OBJ generated by GLM library\n");

    fprintf(file, "# \n");

    fprintf(file, "# GLM library\n");

    fprintf(file, "# Nate Robins\n");

    fprintf(file, "# ndr@pobox.com\n");

    fprintf(file, "# http://www.pobox.com/~ndr\n");

    fprintf(file, "# \n");

**if** (mode **&** GLM\_MATERIAL **&&** model**->**mtllibname) {

        fprintf(file, "\nmtllib %s\n\n", model**->**mtllibname);

        glmWriteMTL(model, filename, model**->**mtllibname);

    }

    fprintf(file, "\n");

    fprintf(file, "# %d vertices\n", model**->**numvertices);

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

        fprintf(file, "v %f %f %f\n",

            model**->**vertices[3 **\*** i **+** 0],

            model**->**vertices[3 **\*** i **+** 1],

            model**->**vertices[3 **\*** i **+** 2]);

    }

**if** (mode **&** GLM\_SMOOTH) {

        fprintf(file, "\n");

        fprintf(file, "# %d normals\n", model**->**numnormals);

**for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {

            fprintf(file, "vn %f %f %f\n",

                model**->**normals[3 **\*** i **+** 0],

                model**->**normals[3 **\*** i **+** 1],

                model**->**normals[3 **\*** i **+** 2]);

        }

    } **else** **if** (mode **&** GLM\_FLAT) {

        fprintf(file, "\n");

        fprintf(file, "# %d normals\n", model**->**numfacetnorms);

**for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {

            fprintf(file, "vn %f %f %f\n",

                model**->**facetnorms[3 **\*** i **+** 0],

                model**->**facetnorms[3 **\*** i **+** 1],

                model**->**facetnorms[3 **\*** i **+** 2]);

        }

    }

**if** (mode **&** GLM\_TEXTURE) {

        fprintf(file, "\n");

        fprintf(file, "# %d texcoords\n", model**->**numtexcoords);

**for** (i **=** 1; i **<=** model**->**numtexcoords; i**++**) {

            fprintf(file, "vt %f %f\n",

                model**->**texcoords[2 **\*** i **+** 0],

                model**->**texcoords[2 **\*** i **+** 1]);

        }

    }

    fprintf(file, "\n");

    fprintf(file, "# %d groups\n", model**->**numgroups);

    fprintf(file, "# %d faces (triangles)\n", model**->**numtriangles);

    fprintf(file, "\n");

    group **=** model**->**groups;

**while**(group) {

        fprintf(file, "g %s\n", group**->**name);

**if** (mode **&** GLM\_MATERIAL)

            fprintf(file, "usemtl %s\n", model**->**materials[group**->**material].name);

**for** (i **=** 0; i **<** group**->**numtriangles; i**++**) {

**if** (mode **&** GLM\_SMOOTH **&&** mode **&** GLM\_TEXTURE) {

                fprintf(file, "f %d/%d/%d %d/%d/%d %d/%d/%d\n",

                    T(group**->**triangles[i]).vindices[0],

                    T(group**->**triangles[i]).tindices[0],

                    T(group**->**triangles[i]).nindices[0],

                    T(group**->**triangles[i]).vindices[1],

                    T(group**->**triangles[i]).tindices[1],

                    T(group**->**triangles[i]).nindices[1],

                    T(group**->**triangles[i]).vindices[2],

                    T(group**->**triangles[i]).tindices[2],

                    T(group**->**triangles[i]).nindices[2]);

            } **else** **if** (mode **&** GLM\_FLAT **&&** mode **&** GLM\_TEXTURE) {

                fprintf(file, "f %d/%d %d/%d %d/%d\n",

                    T(group**->**triangles[i]).vindices[0],

                    T(group**->**triangles[i]).findex,

                    T(group**->**triangles[i]).vindices[1],

                    T(group**->**triangles[i]).findex,

                    T(group**->**triangles[i]).vindices[2],

                    T(group**->**triangles[i]).findex);

            } **else** **if** (mode **&** GLM\_TEXTURE) {

                fprintf(file, "f %d/%d %d/%d %d/%d\n",

                    T(group**->**triangles[i]).vindices[0],

                    T(group**->**triangles[i]).tindices[0],

                    T(group**->**triangles[i]).vindices[1],

                    T(group**->**triangles[i]).tindices[1],

                    T(group**->**triangles[i]).vindices[2],

                    T(group**->**triangles[i]).tindices[2]);

            } **else** **if** (mode **&** GLM\_SMOOTH) {

                fprintf(file, "f %d//%d %d//%d %d//%d\n",

                    T(group**->**triangles[i]).vindices[0],

                    T(group**->**triangles[i]).nindices[0],

                    T(group**->**triangles[i]).vindices[1],

                    T(group**->**triangles[i]).nindices[1],

                    T(group**->**triangles[i]).vindices[2],

                    T(group**->**triangles[i]).nindices[2]);

            } **else** **if** (mode **&** GLM\_FLAT) {

                fprintf(file, "f %d//%d %d//%d %d//%d\n",

                    T(group**->**triangles[i]).vindices[0],

                    T(group**->**triangles[i]).findex,

                    T(group**->**triangles[i]).vindices[1],

                    T(group**->**triangles[i]).findex,

                    T(group**->**triangles[i]).vindices[2],

                    T(group**->**triangles[i]).findex);

            } **else** {

                fprintf(file, "f %d %d %d\n",

                    T(group**->**triangles[i]).vindices[0],

                    T(group**->**triangles[i]).vindices[1],

                    T(group**->**triangles[i]).vindices[2]);

            }

        }

        fprintf(file, "\n");

        group **=** group**->**next;

    }

    fclose(file);

}

GLvoid

**glmDraw**(GLMmodel**\*** model, GLuint mode)

{

**static** GLuint i;

**static** GLMgroup**\*** group;

**static** GLMtriangle**\*** triangle;

**static** GLMmaterial**\*** material;

    assert(model);

    assert(model**->**vertices);

**if** (mode **&** GLM\_FLAT **&&** **!**model**->**facetnorms) {

        printf("glmDraw() warning: flat render mode requested "

            "with no facet normals defined.\n");

        mode **&=** **~**GLM\_FLAT;

    }

**if** (mode **&** GLM\_SMOOTH **&&** **!**model**->**normals) {

        printf("glmDraw() warning: smooth render mode requested "

            "with no normals defined.\n");

        mode **&=** **~**GLM\_SMOOTH;

    }

**if** (mode **&** GLM\_TEXTURE **&&** **!**model**->**texcoords) {

        printf("glmDraw() warning: texture render mode requested "

            "with no texture coordinates defined.\n");

        mode **&=** **~**GLM\_TEXTURE;

    }

**if** (mode **&** GLM\_FLAT **&&** mode **&** GLM\_SMOOTH) {

        printf("glmDraw() warning: flat render mode requested "

            "and smooth render mode requested (using smooth).\n");

        mode **&=** **~**GLM\_FLAT;

    }

**if** (mode **&** GLM\_COLOR **&&** **!**model**->**materials) {

        printf("glmDraw() warning: color render mode requested "

            "with no materials defined.\n");

        mode **&=** **~**GLM\_COLOR;

    }

**if** (mode **&** GLM\_MATERIAL **&&** **!**model**->**materials) {

        printf("glmDraw() warning: material render mode requested "

            "with no materials defined.\n");

        mode **&=** **~**GLM\_MATERIAL;

    }

**if** (mode **&** GLM\_COLOR **&&** mode **&** GLM\_MATERIAL) {

        printf("glmDraw() warning: color and material render mode requested "

            "using only material mode.\n");

        mode **&=** **~**GLM\_COLOR;

    }

**if** (mode **&** GLM\_COLOR)

        glEnable(GL\_COLOR\_MATERIAL);

**else** **if** (mode **&** GLM\_MATERIAL)

        glDisable(GL\_COLOR\_MATERIAL);

    group **=** model**->**groups;

**while** (group) {

**if** (mode **&** GLM\_MATERIAL) {

            material **=** **&**model**->**materials[group**->**material];

            glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, material**->**ambient);

            glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, material**->**diffuse);

            glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, material**->**specular);

            glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, material**->**shininess);

        }

**if** (mode **&** GLM\_COLOR) {

            glColor3fv(material**->**diffuse);

        }

        glBegin(GL\_TRIANGLES);

**for** (i **=** 0; i **<** group**->**numtriangles; i**++**) {

            triangle **=** **&**T(group**->**triangles[i]);

**if** (mode **&** GLM\_FLAT)

                glNormal3fv(**&**model**->**facetnorms[3 **\*** triangle**->**findex]);

**if** (mode **&** GLM\_SMOOTH)

                glNormal3fv(**&**model**->**normals[3 **\*** triangle**->**nindices[0]]);

**if** (mode **&** GLM\_TEXTURE)

                glTexCoord2fv(**&**model**->**texcoords[2 **\*** triangle**->**tindices[0]]);

            glVertex3fv(**&**model**->**vertices[3 **\*** triangle**->**vindices[0]]);

**if** (mode **&** GLM\_SMOOTH)

                glNormal3fv(**&**model**->**normals[3 **\*** triangle**->**nindices[1]]);

**if** (mode **&** GLM\_TEXTURE)

                glTexCoord2fv(**&**model**->**texcoords[2 **\*** triangle**->**tindices[1]]);

            glVertex3fv(**&**model**->**vertices[3 **\*** triangle**->**vindices[1]]);

**if** (mode **&** GLM\_SMOOTH)

                glNormal3fv(**&**model**->**normals[3 **\*** triangle**->**nindices[2]]);

**if** (mode **&** GLM\_TEXTURE)

                glTexCoord2fv(**&**model**->**texcoords[2 **\*** triangle**->**tindices[2]]);

            glVertex3fv(**&**model**->**vertices[3 **\*** triangle**->**vindices[2]]);

        }

        glEnd();

        group **=** group**->**next;

    }

}

GLuint

**glmList**(GLMmodel**\*** model, GLuint mode)

{

    GLuint list;

    list **=** glGenLists(1);

    glNewList(list, GL\_COMPILE);

    glmDraw(model, mode);

    glEndList();

**return** list;

}

GLvoid

**glmWeld**(GLMmodel**\*** model, GLfloat epsilon)

{

    GLfloat**\*** vectors;

    GLfloat**\*** copies;

    GLuint numvectors;

    GLuint i;

    numvectors **=** model**->**numvertices;

    vectors **=** model**->**vertices;

    copies **=** glmWeldVectors(vectors, **&**numvectors, epsilon);

**#if 0**

*printf("glmWeld(): %d redundant vertices.\n",*

*model->numvertices - numvectors - 1);*

**#endif**

**for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {

        T(i).vindices[0] **=** (GLuint)vectors[3 **\*** T(i).vindices[0] **+** 0];

        T(i).vindices[1] **=** (GLuint)vectors[3 **\*** T(i).vindices[1] **+** 0];

        T(i).vindices[2] **=** (GLuint)vectors[3 **\*** T(i).vindices[2] **+** 0];

    }

    free(vectors);

    model**->**numvertices **=** numvectors;

    model**->**vertices **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***

        3 **\*** (model**->**numvertices **+** 1));

**for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {

        model**->**vertices[3 **\*** i **+** 0] **=** copies[3 **\*** i **+** 0];

        model**->**vertices[3 **\*** i **+** 1] **=** copies[3 **\*** i **+** 1];

        model**->**vertices[3 **\*** i **+** 2] **=** copies[3 **\*** i **+** 2];

    }

    free(copies);

}

GLubyte**\***

**glmReadPPM**(**char\*** filename, **int\*** width, **int\*** height)

{

**FILE\*** fp;

**int** i, w, h, d;

**unsigned** **char\*** image;

**char** head[70];

    fp **=** fopen(filename, "rb");

**if** (**!**fp) {

        perror(filename);

**return** NULL;

    }

    fgets(head, 70, fp);

**if** (strncmp(head, "P6", 2)) {

        fprintf(stderr, "%s: Not a raw PPM file\n", filename);

**return** NULL;

    }

    i **=** 0;

**while**(i **<** 3) {

        fgets(head, 70, fp);

**if** (head[0] **==** '#')

**continue**;

**if** (i **==** 0)

            i **+=** sscanf(head, "%d %d %d", **&**w, **&**h, **&**d);

**else** **if** (i **==** 1)

            i **+=** sscanf(head, "%d %d", **&**h, **&**d);

**else** **if** (i **==** 2)

            i **+=** sscanf(head, "%d", **&**d);

    }

    image **=** (**unsigned** **char\***)malloc(**sizeof**(**unsigned** **char**)**\***w**\***h**\***3);

    fread(image, **sizeof**(**unsigned** **char**), w**\***h**\***3, fp);

    fclose(fp);

**\***width **=** w;

**\***height **=** h;

**return** image;

}

**#if 0**

*if (model->numnormals) {*

*numvectors = model->numnormals;*

*vectors = model->normals;*

*copies = glmOptimizeVectors(vectors, &numvectors);*

*printf("glmOptimize(): %d redundant normals.\n",*

*model->numnormals - numvectors);*

*for (i = 0; i < model->numtriangles; i++) {*

*T(i).nindices[0] = (GLuint)vectors[3 \* T(i).nindices[0] + 0];*

*T(i).nindices[1] = (GLuint)vectors[3 \* T(i).nindices[1] + 0];*

*T(i).nindices[2] = (GLuint)vectors[3 \* T(i).nindices[2] + 0];*

*}*

*model->numnormals = numvectors;*

*model->normals = (GLfloat\*)malloc(sizeof(GLfloat) \**

*3 \* (model->numnormals + 1));*

*for (i = 1; i <= model->numnormals; i++) {*

*model->normals[3 \* i + 0] = copies[3 \* i + 0];*

*model->normals[3 \* i + 1] = copies[3 \* i + 1];*

*model->normals[3 \* i + 2] = copies[3 \* i + 2];*

*}*

*free(copies);*

*}*

*if (model->numtexcoords) {*

*numvectors = model->numtexcoords;*

*vectors = model->texcoords;*

*copies = glmOptimizeVectors(vectors, &numvectors);*

*printf("glmOptimize(): %d redundant texcoords.\n",*

*model->numtexcoords - numvectors);*

*for (i = 0; i < model->numtriangles; i++) {*

*for (j = 0; j < 3; j++) {*

*T(i).tindices[j] = (GLuint)vectors[3 \* T(i).tindices[j] + 0];*

*}*

*}*

*free(vectors);*

*model->numtexcoords = numvectors;*

*model->texcoords = (GLfloat\*)malloc(sizeof(GLfloat) \**

*2 \* (model->numtexcoords + 1));*

*for (i = 1; i <= model->numtexcoords; i++) {*

*model->texcoords[2 \* i + 0] = copies[2 \* i + 0];*

*model->texcoords[2 \* i + 1] = copies[2 \* i + 1];*

*}*

*free(copies);*

*}*

**#endif**

**#if 0**

*for (i = 1; i <= model->numvertices; i++) {*

*for (j = 0; j < model->numtriangles; i++) {*

*if (T(j).vindices[0] == i ||*

*T(j).vindices[1] == i ||*

*T(j).vindices[1] == i)*

*break;*

*}*

*}*

**#endif**

**BAB III**

**IMPLEMENTASI PROGRAM**

Sourcecode Program OpenGL :

|  |
| --- |
| **glm.h**  **#if defined(\_\_APPLE\_\_) || defined(MACOSX)**  **#include <GLUT/glut.h>**  **#else**  **#include <GL/glut.h>**  **#endif**  **#ifndef M\_PI**  **#define M\_PI 3.14159265f**  **#endif**  **#define GLM\_NONE (0)**  **#define GLM\_FLAT (1 << 0)**  **#define GLM\_SMOOTH (1 << 1)**  **#define GLM\_TEXTURE (1 << 2)**  **#define GLM\_COLOR (1 << 3)**  **#define GLM\_MATERIAL (1 << 4)**  **typedef** **struct** \_GLMmaterial  {  **char\*** name;    GLfloat diffuse[4];    GLfloat ambient[4];    GLfloat specular[4];    GLfloat emmissive[4];    GLfloat shininess;  } GLMmaterial;  **typedef** **struct** \_GLMtriangle {    GLuint vindices[3];    GLuint nindices[3];    GLuint tindices[3];    GLuint findex;  } GLMtriangle;  **typedef** **struct** \_GLMgroup {  **char\*** name;    GLuint numtriangles;    GLuint**\*** triangles;    GLuint material;  **struct** \_GLMgroup**\*** next;  } GLMgroup;  **typedef** **struct** \_GLMmodel {  **char\*** pathname;  **char\*** mtllibname;    GLuint numvertices;    GLfloat**\*** vertices;    GLuint numnormals;    GLfloat**\*** normals;    GLuint numtexcoords;    GLfloat**\*** texcoords;    GLuint numfacetnorms;    GLfloat**\*** facetnorms;    GLuint numtriangles;    GLMtriangle**\*** triangles;    GLuint nummaterials;    GLMmaterial**\*** materials;    GLuint numgroups;    GLMgroup**\*** groups;    GLfloat position[3];  } GLMmodel;  GLfloat  **glmUnitize**(GLMmodel**\*** model);  GLvoid  **glmDimensions**(GLMmodel**\*** model, GLfloat**\*** dimensions);  GLvoid  **glmScale**(GLMmodel**\*** model, GLfloat scale);  GLvoid  **glmReverseWinding**(GLMmodel**\*** model);  GLvoid  **glmFacetNormals**(GLMmodel**\*** model);  GLvoid  **glmVertexNormals**(GLMmodel**\*** model, GLfloat angle);  GLvoid  **glmLinearTexture**(GLMmodel**\*** model);  GLvoid  **glmSpheremapTexture**(GLMmodel**\*** model);  GLvoid  **glmDelete**(GLMmodel**\*** model);  GLMmodel**\***  **glmReadOBJ**(**char\*** filename);  GLvoid  **glmWriteOBJ**(GLMmodel**\*** model, **char\*** filename, GLuint mode);  GLvoid  **glmDraw**(GLMmodel**\*** model, GLuint mode);  GLuint  **glmList**(GLMmodel**\*** model, GLuint mode);  GLvoid  **glmWeld**(GLMmodel**\*** model, GLfloat epsilon);  GLubyte**\***  **glmReadPPM**(**char\*** filename, **int\*** width, **int\*** height);  **mac.cpp**  **#include <math.h>**  **#include <stdio.h>**  **#include <stdlib.h>**  **#include <string.h>**  **#include <assert.h>**  **#include "glm.h"**  **#define T(x) (model->triangles[(x)])**  **typedef** **struct** \_GLMnode {      GLuint index;      GLboolean averaged;  **struct** \_GLMnode**\*** next;  } GLMnode;  **static** GLfloat  **glmMax**(GLfloat a, GLfloat b)  {  **if** (b **>** a)  **return** b;  **return** a;  }  **static** GLfloat  **glmAbs**(GLfloat f)  {  **if** (f **<** 0)  **return** **-**f;  **return** f;  }  **static** GLfloat  **glmDot**(GLfloat**\*** u, GLfloat**\*** v)  {      assert(u); assert(v);    **return** u[0]**\***v[0] **+** u[1]**\***v[1] **+** u[2]**\***v[2];  }  **static** GLvoid  **glmCross**(GLfloat**\*** u, GLfloat**\*** v, GLfloat**\*** n)  {      assert(u); assert(v); assert(n);        n[0] **=** u[1]**\***v[2] **-** u[2]**\***v[1];      n[1] **=** u[2]**\***v[0] **-** u[0]**\***v[2];      n[2] **=** u[0]**\***v[1] **-** u[1]**\***v[0];  }  **static** GLvoid  **glmNormalize**(GLfloat**\*** v)  {      GLfloat l;        assert(v);        l **=** (GLfloat)sqrt(v[0]**\***v[0] **+** v[1]**\***v[1] **+** v[2]**\***v[2]);      v[0] **/=** l;      v[1] **/=** l;      v[2] **/=** l;  }  **static** GLboolean  **glmEqual**(GLfloat**\*** u, GLfloat**\*** v, GLfloat epsilon)  {  **if** (glmAbs(u[0] **-** v[0]) **<** epsilon **&&**          glmAbs(u[1] **-** v[1]) **<** epsilon **&&**          glmAbs(u[2] **-** v[2]) **<** epsilon)      {  **return** GL\_TRUE;      }  **return** GL\_FALSE;  }  GLfloat**\***  **glmWeldVectors**(GLfloat**\*** vectors, GLuint**\*** numvectors, GLfloat epsilon)  {      GLfloat**\*** copies;      GLuint copied;      GLuint i, j;        copies **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\*** 3 **\*** (**\***numvectors **+** 1));      memcpy(copies, vectors, (**sizeof**(GLfloat) **\*** 3 **\*** (**\***numvectors **+** 1)));        copied **=** 1;  **for** (i **=** 1; i **<=** **\***numvectors; i**++**) {  **for** (j **=** 1; j **<=** copied; j**++**) {  **if** (glmEqual(**&**vectors[3 **\*** i], **&**copies[3 **\*** j], epsilon)) {  **goto** duplicate;              }          }            copies[3 **\*** copied **+** 0] **=** vectors[3 **\*** i **+** 0];          copies[3 **\*** copied **+** 1] **=** vectors[3 **\*** i **+** 1];          copies[3 **\*** copied **+** 2] **=** vectors[3 **\*** i **+** 2];          j **=** copied;          copied**++**;    duplicate:          vectors[3 **\*** i **+** 0] **=** (GLfloat)j;      }    **\***numvectors **=** copied**-**1;  **return** copies;  }  GLMgroup**\***  **glmFindGroup**(GLMmodel**\*** model, **char\*** name)  {      GLMgroup**\*** group;        assert(model);        group **=** model**->**groups;  **while**(group) {  **if** (**!**strcmp(name, group**->**name))  **break**;          group **=** group**->**next;      }    **return** group;  }  GLMgroup**\***  **glmAddGroup**(GLMmodel**\*** model, **char\*** name)  {      GLMgroup**\*** group;        group **=** glmFindGroup(model, name);  **if** (**!**group) {          group **=** (GLMgroup**\***)malloc(**sizeof**(GLMgroup));          group**->**name **=** strdup(name);          group**->**material **=** 0;          group**->**numtriangles **=** 0;          group**->**triangles **=** NULL;          group**->**next **=** model**->**groups;          model**->**groups **=** group;          model**->**numgroups**++**;      }    **return** group;  }  GLuint  **glmFindMaterial**(GLMmodel**\*** model, **char\*** name)  {      GLuint i;    **for** (i **=** 0; i **<** model**->**nummaterials; i**++**) {  **if** (**!**strcmp(model**->**materials[i].name, name))  **goto** found;      }        printf("glmFindMaterial(): can't find material \"%s\".\n", name);      i **=** 0;    found:  **return** i;  }  **static** **char\***  **glmDirName**(**char\*** path)  {  **char\*** dir;  **char\*** s;        dir **=** strdup(path);        s **=** strrchr(dir, '/');  **if** (s)          s[1] **=** '\0';  **else**          dir[0] **=** '\0';    **return** dir;  }  **static** GLvoid  **glmReadMTL**(GLMmodel**\*** model, **char\*** name)  {  **FILE\*** file;  **char\*** dir;  **char\*** filename;  **char** buf[128];      GLuint nummaterials, i;        dir **=** glmDirName(model**->**pathname);      filename **=** (**char\***)malloc(**sizeof**(**char**) **\*** (strlen(dir) **+** strlen(name) **+** 1));      strcpy(filename, dir);      strcat(filename, name);      free(dir);        file **=** fopen(filename, "r");  **if** (**!**file) {          fprintf(stderr, "glmReadMTL() failed: can't open material file \"%s\".\n",              filename);          exit(1);      }      free(filename);        nummaterials **=** 1;  **while**(fscanf(file, "%s", buf) **!=** EOF) {  **switch**(buf[0]) {  **case** '#':              fgets(buf, **sizeof**(buf), file);  **break**;  **case** 'n':              fgets(buf, **sizeof**(buf), file);              nummaterials**++**;              sscanf(buf, "%s %s", buf, buf);  **break**;          default:              fgets(buf, **sizeof**(buf), file);  **break**;          }      }        rewind(file);        model**->**materials **=** (GLMmaterial**\***)malloc(**sizeof**(GLMmaterial) **\*** nummaterials);      model**->**nummaterials **=** nummaterials;      **for** (i **=** 0; i **<** nummaterials; i**++**) {          model**->**materials[i].name **=** NULL;          model**->**materials[i].shininess **=** 65.0;          model**->**materials[i].diffuse[0] **=** 0.8;          model**->**materials[i].diffuse[1] **=** 0.8;          model**->**materials[i].diffuse[2] **=** 0.8;          model**->**materials[i].diffuse[3] **=** 1.0;          model**->**materials[i].ambient[0] **=** 0.2;          model**->**materials[i].ambient[1] **=** 0.2;          model**->**materials[i].ambient[2] **=** 0.2;          model**->**materials[i].ambient[3] **=** 1.0;          model**->**materials[i].specular[0] **=** 0.0;          model**->**materials[i].specular[1] **=** 0.0;          model**->**materials[i].specular[2] **=** 0.0;          model**->**materials[i].specular[3] **=** 1.0;      }      model**->**materials[0].name **=** strdup("default");        nummaterials **=** 0;  **while**(fscanf(file, "%s", buf) **!=** EOF) {  **switch**(buf[0]) {  **case** '#':              fgets(buf, **sizeof**(buf), file);  **break**;  **case** 'n':              fgets(buf, **sizeof**(buf), file);              sscanf(buf, "%s %s", buf, buf);              nummaterials**++**;              model**->**materials[nummaterials].name **=** strdup(buf);  **break**;  **case** 'N':              fscanf(file, "%f", **&**model**->**materials[nummaterials].shininess);              model**->**materials[nummaterials].shininess **/=** 1000.0;              model**->**materials[nummaterials].shininess **\*=** 128.0;  **break**;  **case** 'K':  **switch**(buf[1]) {  **case** 'd':                  fscanf(file, "%f %f %f",  **&**model**->**materials[nummaterials].diffuse[0],  **&**model**->**materials[nummaterials].diffuse[1],  **&**model**->**materials[nummaterials].diffuse[2]);  **break**;  **case** 's':                  fscanf(file, "%f %f %f",  **&**model**->**materials[nummaterials].specular[0],  **&**model**->**materials[nummaterials].specular[1],  **&**model**->**materials[nummaterials].specular[2]);  **break**;  **case** 'a':                  fscanf(file, "%f %f %f",  **&**model**->**materials[nummaterials].ambient[0],  **&**model**->**materials[nummaterials].ambient[1],  **&**model**->**materials[nummaterials].ambient[2]);  **break**;              default:                  fgets(buf, **sizeof**(buf), file);  **break**;              }  **break**;              default:                  fgets(buf, **sizeof**(buf), file);  **break**;          }      }  }  **static** GLvoid  **glmWriteMTL**(GLMmodel**\*** model, **char\*** modelpath, **char\*** mtllibname)  {  **FILE\*** file;  **char\*** dir;  **char\*** filename;      GLMmaterial**\*** material;      GLuint i;        dir **=** glmDirName(modelpath);      filename **=** (**char\***)malloc(**sizeof**(**char**) **\*** (strlen(dir)**+**strlen(mtllibname)));      strcpy(filename, dir);      strcat(filename, mtllibname);      free(dir);        file **=** fopen(filename, "w");  **if** (**!**file) {          fprintf(stderr, "glmWriteMTL() failed: can't open file \"%s\".\n",              filename);          exit(1);      }      free(filename);        fprintf(file, "# \n");      fprintf(file, "# Wavefront MTL generated by GLM library\n");      fprintf(file, "# \n");      fprintf(file, "# GLM library\n");      fprintf(file, "# Nate Robins\n");      fprintf(file, "# ndr@pobox.com\n");      fprintf(file, "# http://www.pobox.com/~ndr\n");      fprintf(file, "# \n\n");    **for** (i **=** 0; i **<** model**->**nummaterials; i**++**) {          material **=** **&**model**->**materials[i];          fprintf(file, "newmtl %s\n", material**->**name);          fprintf(file, "Ka %f %f %f\n",              material**->**ambient[0], material**->**ambient[1], material**->**ambient[2]);          fprintf(file, "Kd %f %f %f\n",              material**->**diffuse[0], material**->**diffuse[1], material**->**diffuse[2]);          fprintf(file, "Ks %f %f %f\n",              material**->**specular[0],material**->**specular[1],material**->**specular[2]);          fprintf(file, "Ns %f\n", material**->**shininess **/** 128.0 **\*** 1000.0);          fprintf(file, "\n");      }  }  **static** GLvoid  **glmFirstPass**(GLMmodel**\*** model, **FILE\*** file)  {      GLuint numvertices;      GLuint numnormals;      GLuint numtexcoords;      GLuint numtriangles;      GLMgroup**\*** group;  **int** v, n, t;  **char** buf[128];        group **=** glmAddGroup(model, "default");        numvertices **=** numnormals **=** numtexcoords **=** numtriangles **=** 0;  **while**(fscanf(file, "%s", buf) **!=** EOF) {  **switch**(buf[0]) {  **case** '#':              fgets(buf, **sizeof**(buf), file);  **break**;  **case** 'v':  **switch**(buf[1]) {  **case** '\0':                  fgets(buf, **sizeof**(buf), file);                  numvertices**++**;  **break**;  **case** 'n':                  fgets(buf, **sizeof**(buf), file);                  numnormals**++**;  **break**;  **case** 't':                  fgets(buf, **sizeof**(buf), file);                  numtexcoords**++**;  **break**;              default:                  printf("glmFirstPass(): Unknown token \"%s\".\n", buf);                  exit(1);  **break**;              }  **break**;  **case** 'm':                  fgets(buf, **sizeof**(buf), file);                  sscanf(buf, "%s %s", buf, buf);                  model**->**mtllibname **=** strdup(buf);                  glmReadMTL(model, buf);  **break**;  **case** 'u':                  fgets(buf, **sizeof**(buf), file);  **break**;  **case** 'g':                  fgets(buf, **sizeof**(buf), file);  **#if SINGLE\_STRING\_GROUP\_NAMES**                  sscanf(buf, "%s", buf);  **#else**                  buf[strlen(buf)**-**1] **=** '\0';  **#endif**                  group **=** glmAddGroup(model, buf);  **break**;  **case** 'f':                  v **=** n **=** t **=** 0;                  fscanf(file, "%s", buf);  **if** (strstr(buf, "//")) {                      sscanf(buf, "%d//%d", **&**v, **&**n);                      fscanf(file, "%d//%d", **&**v, **&**n);                      fscanf(file, "%d//%d", **&**v, **&**n);                      numtriangles**++**;                      group**->**numtriangles**++**;  **while**(fscanf(file, "%d//%d", **&**v, **&**n) **>** 0) {                          numtriangles**++**;                          group**->**numtriangles**++**;                      }                  } **else** **if** (sscanf(buf, "%d/%d/%d", **&**v, **&**t, **&**n) **==** 3) {                      fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);                      fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);                      numtriangles**++**;                      group**->**numtriangles**++**;  **while**(fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n) **>** 0) {                          numtriangles**++**;                          group**->**numtriangles**++**;                      }                  } **else** **if** (sscanf(buf, "%d/%d", **&**v, **&**t) **==** 2) {                      fscanf(file, "%d/%d", **&**v, **&**t);                      fscanf(file, "%d/%d", **&**v, **&**t);                      numtriangles**++**;                      group**->**numtriangles**++**;  **while**(fscanf(file, "%d/%d", **&**v, **&**t) **>** 0) {                          numtriangles**++**;                          group**->**numtriangles**++**;                      }                  } **else** {                      fscanf(file, "%d", **&**v);                      fscanf(file, "%d", **&**v);                      numtriangles**++**;                      group**->**numtriangles**++**;  **while**(fscanf(file, "%d", **&**v) **>** 0) {                          numtriangles**++**;                          group**->**numtriangles**++**;                      }                  }  **break**;                default:                  fgets(buf, **sizeof**(buf), file);  **break**;          }    }      model**->**numvertices **=** numvertices;    model**->**numnormals **=** numnormals;    model**->**numtexcoords **=** numtexcoords;    model**->**numtriangles **=** numtriangles;      group **=** model**->**groups;  **while**(group) {        group**->**triangles **=** (GLuint**\***)malloc(**sizeof**(GLuint) **\*** group**->**numtriangles);        group**->**numtriangles **=** 0;        group **=** group**->**next;    }  }  **static** GLvoid  **glmSecondPass**(GLMmodel**\*** model, **FILE\*** file)  {      GLuint numvertices;      GLuint numnormals;      GLuint numtexcoords;      GLuint numtriangles;      GLfloat**\*** vertices;      GLfloat**\*** normals;      GLfloat**\*** texcoords;      GLMgroup**\*** group;      GLuint material;  **int** v, n, t;  **char** buf[128];        vertices **=** model**->**vertices;      normals **=** model**->**normals;      texcoords **=** model**->**texcoords;      group **=** model**->**groups;        numvertices **=** numnormals **=** numtexcoords **=** 1;      numtriangles **=** 0;      material **=** 0;  **while**(fscanf(file, "%s", buf) **!=** EOF) {  **switch**(buf[0]) {  **case** '#':              fgets(buf, **sizeof**(buf), file);  **break**;  **case** 'v':  **switch**(buf[1]) {  **case** '\0':                  fscanf(file, "%f %f %f",  **&**vertices[3 **\*** numvertices **+** 0],  **&**vertices[3 **\*** numvertices **+** 1],  **&**vertices[3 **\*** numvertices **+** 2]);                  numvertices**++**;  **break**;  **case** 'n':                  fscanf(file, "%f %f %f",  **&**normals[3 **\*** numnormals **+** 0],  **&**normals[3 **\*** numnormals **+** 1],  **&**normals[3 **\*** numnormals **+** 2]);                  numnormals**++**;  **break**;  **case** 't':                  fscanf(file, "%f %f",  **&**texcoords[2 **\*** numtexcoords **+** 0],  **&**texcoords[2 **\*** numtexcoords **+** 1]);                  numtexcoords**++**;  **break**;              }  **break**;  **case** 'u':                  fgets(buf, **sizeof**(buf), file);                  sscanf(buf, "%s %s", buf, buf);                  group**->**material **=** material **=** glmFindMaterial(model, buf);  **break**;  **case** 'g':                  fgets(buf, **sizeof**(buf), file);  **#if SINGLE\_STRING\_GROUP\_NAMES**                  sscanf(buf, "%s", buf);  **#else**                  buf[strlen(buf)**-**1] **=** '\0';  **#endif**                  group **=** glmFindGroup(model, buf);                  group**->**material **=** material;  **break**;  **case** 'f':                  v **=** n **=** t **=** 0;                  fscanf(file, "%s", buf);  **if** (strstr(buf, "//")) {                        sscanf(buf, "%d//%d", **&**v, **&**n);                      T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).nindices[0] **=** n **<** 0 **?** n **+** numnormals **:** n;                      fscanf(file, "%d//%d", **&**v, **&**n);                      T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).nindices[1] **=** n **<** 0 **?** n **+** numnormals **:** n;                      fscanf(file, "%d//%d", **&**v, **&**n);                      T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;                      group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                      numtriangles**++**;  **while**(fscanf(file, "%d//%d", **&**v, **&**n) **>** 0) {                          T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];                          T(numtriangles).nindices[0] **=** T(numtriangles**-**1).nindices[0];                          T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];                          T(numtriangles).nindices[1] **=** T(numtriangles**-**1).nindices[2];                          T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                          T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;                          group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                          numtriangles**++**;                      }                  } **else** **if** (sscanf(buf, "%d/%d/%d", **&**v, **&**t, **&**n) **==** 3) {                        T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).tindices[0] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                      T(numtriangles).nindices[0] **=** n **<** 0 **?** n **+** numnormals **:** n;                      fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);                      T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).tindices[1] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                      T(numtriangles).nindices[1] **=** n **<** 0 **?** n **+** numnormals **:** n;                      fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n);                      T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                      T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;                      group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                      numtriangles**++**;  **while**(fscanf(file, "%d/%d/%d", **&**v, **&**t, **&**n) **>** 0) {                          T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];                          T(numtriangles).tindices[0] **=** T(numtriangles**-**1).tindices[0];                          T(numtriangles).nindices[0] **=** T(numtriangles**-**1).nindices[0];                          T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];                          T(numtriangles).tindices[1] **=** T(numtriangles**-**1).tindices[2];                          T(numtriangles).nindices[1] **=** T(numtriangles**-**1).nindices[2];                          T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                          T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                          T(numtriangles).nindices[2] **=** n **<** 0 **?** n **+** numnormals **:** n;                          group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                          numtriangles**++**;                      }                  } **else** **if** (sscanf(buf, "%d/%d", **&**v, **&**t) **==** 2) {                        T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).tindices[0] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                      fscanf(file, "%d/%d", **&**v, **&**t);                      T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).tindices[1] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                      fscanf(file, "%d/%d", **&**v, **&**t);                      T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                      T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                      group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                      numtriangles**++**;  **while**(fscanf(file, "%d/%d", **&**v, **&**t) **>** 0) {                          T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];                          T(numtriangles).tindices[0] **=** T(numtriangles**-**1).tindices[0];                          T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];                          T(numtriangles).tindices[1] **=** T(numtriangles**-**1).tindices[2];                          T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                          T(numtriangles).tindices[2] **=** t **<** 0 **?** t **+** numtexcoords **:** t;                          group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                          numtriangles**++**;                      }                  } **else** {                        sscanf(buf, "%d", **&**v);                      T(numtriangles).vindices[0] **=** v **<** 0 **?** v **+** numvertices **:** v;                      fscanf(file, "%d", **&**v);                      T(numtriangles).vindices[1] **=** v **<** 0 **?** v **+** numvertices **:** v;                      fscanf(file, "%d", **&**v);                      T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                      group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                      numtriangles**++**;  **while**(fscanf(file, "%d", **&**v) **>** 0) {                          T(numtriangles).vindices[0] **=** T(numtriangles**-**1).vindices[0];                          T(numtriangles).vindices[1] **=** T(numtriangles**-**1).vindices[2];                          T(numtriangles).vindices[2] **=** v **<** 0 **?** v **+** numvertices **:** v;                          group**->**triangles[group**->**numtriangles**++**] **=** numtriangles;                          numtriangles**++**;                      }                  }  **break**;                default:                    fgets(buf, **sizeof**(buf), file);  **break**;      }    }    **#if 0**  *printf(" Memory: %d bytes\n",*  *numvertices \* 3\*sizeof(GLfloat) +*  *numnormals \* 3\*sizeof(GLfloat) \* (numnormals ? 1 : 0) +*  *numtexcoords \* 3\*sizeof(GLfloat) \* (numtexcoords ? 1 : 0) +*  *numtriangles \* sizeof(GLMtriangle));*  **#endif**  }  GLfloat  **glmUnitize**(GLMmodel**\*** model)  {      GLuint i;      GLfloat maxx, minx, maxy, miny, maxz, minz;      GLfloat cx, cy, cz, w, h, d;      GLfloat scale;        assert(model);      assert(model**->**vertices);          maxx **=** minx **=** model**->**vertices[3 **+** 0];      maxy **=** miny **=** model**->**vertices[3 **+** 1];      maxz **=** minz **=** model**->**vertices[3 **+** 2];  **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {  **if** (maxx **<** model**->**vertices[3 **\*** i **+** 0])              maxx **=** model**->**vertices[3 **\*** i **+** 0];  **if** (minx **>** model**->**vertices[3 **\*** i **+** 0])              minx **=** model**->**vertices[3 **\*** i **+** 0];    **if** (maxy **<** model**->**vertices[3 **\*** i **+** 1])              maxy **=** model**->**vertices[3 **\*** i **+** 1];  **if** (miny **>** model**->**vertices[3 **\*** i **+** 1])              miny **=** model**->**vertices[3 **\*** i **+** 1];    **if** (maxz **<** model**->**vertices[3 **\*** i **+** 2])              maxz **=** model**->**vertices[3 **\*** i **+** 2];  **if** (minz **>** model**->**vertices[3 **\*** i **+** 2])              minz **=** model**->**vertices[3 **\*** i **+** 2];      }          w **=** glmAbs(maxx) **+** glmAbs(minx);      h **=** glmAbs(maxy) **+** glmAbs(miny);      d **=** glmAbs(maxz) **+** glmAbs(minz);          cx **=** (maxx **+** minx) **/** 2.0;      cy **=** (maxy **+** miny) **/** 2.0;      cz **=** (maxz **+** minz) **/** 2.0;          scale **=** 2.0 **/** glmMax(glmMax(w, h), d);      **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {          model**->**vertices[3 **\*** i **+** 0] **-=** cx;          model**->**vertices[3 **\*** i **+** 1] **-=** cy;          model**->**vertices[3 **\*** i **+** 2] **-=** cz;          model**->**vertices[3 **\*** i **+** 0] **\*=** scale;          model**->**vertices[3 **\*** i **+** 1] **\*=** scale;          model**->**vertices[3 **\*** i **+** 2] **\*=** scale;      }    **return** scale;  }  GLvoid  **glmDimensions**(GLMmodel**\*** model, GLfloat**\*** dimensions)  {      GLuint i;      GLfloat maxx, minx, maxy, miny, maxz, minz;        assert(model);      assert(model**->**vertices);      assert(dimensions);          maxx **=** minx **=** model**->**vertices[3 **+** 0];      maxy **=** miny **=** model**->**vertices[3 **+** 1];      maxz **=** minz **=** model**->**vertices[3 **+** 2];  **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {  **if** (maxx **<** model**->**vertices[3 **\*** i **+** 0])              maxx **=** model**->**vertices[3 **\*** i **+** 0];  **if** (minx **>** model**->**vertices[3 **\*** i **+** 0])              minx **=** model**->**vertices[3 **\*** i **+** 0];    **if** (maxy **<** model**->**vertices[3 **\*** i **+** 1])              maxy **=** model**->**vertices[3 **\*** i **+** 1];  **if** (miny **>** model**->**vertices[3 **\*** i **+** 1])              miny **=** model**->**vertices[3 **\*** i **+** 1];    **if** (maxz **<** model**->**vertices[3 **\*** i **+** 2])              maxz **=** model**->**vertices[3 **\*** i **+** 2];  **if** (minz **>** model**->**vertices[3 **\*** i **+** 2])              minz **=** model**->**vertices[3 **\*** i **+** 2];      }          dimensions[0] **=** glmAbs(maxx) **+** glmAbs(minx);      dimensions[1] **=** glmAbs(maxy) **+** glmAbs(miny);      dimensions[2] **=** glmAbs(maxz) **+** glmAbs(minz);  }  GLvoid  **glmScale**(GLMmodel**\*** model, GLfloat scale)  {      GLuint i;    **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {          model**->**vertices[3 **\*** i **+** 0] **\*=** scale;          model**->**vertices[3 **\*** i **+** 1] **\*=** scale;          model**->**vertices[3 **\*** i **+** 2] **\*=** scale;      }  }  GLvoid  **glmReverseWinding**(GLMmodel**\*** model)  {      GLuint i, swap;        assert(model);    **for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {          swap **=** T(i).vindices[0];          T(i).vindices[0] **=** T(i).vindices[2];          T(i).vindices[2] **=** swap;    **if** (model**->**numnormals) {              swap **=** T(i).nindices[0];              T(i).nindices[0] **=** T(i).nindices[2];              T(i).nindices[2] **=** swap;          }    **if** (model**->**numtexcoords) {              swap **=** T(i).tindices[0];              T(i).tindices[0] **=** T(i).tindices[2];              T(i).tindices[2] **=** swap;          }      }      **for** (i **=** 1; i **<=** model**->**numfacetnorms; i**++**) {          model**->**facetnorms[3 **\*** i **+** 0] **=** **-**model**->**facetnorms[3 **\*** i **+** 0];          model**->**facetnorms[3 **\*** i **+** 1] **=** **-**model**->**facetnorms[3 **\*** i **+** 1];          model**->**facetnorms[3 **\*** i **+** 2] **=** **-**model**->**facetnorms[3 **\*** i **+** 2];      }    **for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {          model**->**normals[3 **\*** i **+** 0] **=** **-**model**->**normals[3 **\*** i **+** 0];          model**->**normals[3 **\*** i **+** 1] **=** **-**model**->**normals[3 **\*** i **+** 1];          model**->**normals[3 **\*** i **+** 2] **=** **-**model**->**normals[3 **\*** i **+** 2];      }  }  GLvoid  **glmFacetNormals**(GLMmodel**\*** model)  {      GLuint i;      GLfloat u[3];      GLfloat v[3];        assert(model);      assert(model**->**vertices);      **if** (model**->**facetnorms)          free(model**->**facetnorms);          model**->**numfacetnorms **=** model**->**numtriangles;      model**->**facetnorms **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***                         3 **\*** (model**->**numfacetnorms **+** 1));    **for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {          model**->**triangles[i].findex **=** i**+**1;            u[0] **=** model**->**vertices[3 **\*** T(i).vindices[1] **+** 0] **-**              model**->**vertices[3 **\*** T(i).vindices[0] **+** 0];          u[1] **=** model**->**vertices[3 **\*** T(i).vindices[1] **+** 1] **-**              model**->**vertices[3 **\*** T(i).vindices[0] **+** 1];          u[2] **=** model**->**vertices[3 **\*** T(i).vindices[1] **+** 2] **-**              model**->**vertices[3 **\*** T(i).vindices[0] **+** 2];            v[0] **=** model**->**vertices[3 **\*** T(i).vindices[2] **+** 0] **-**              model**->**vertices[3 **\*** T(i).vindices[0] **+** 0];          v[1] **=** model**->**vertices[3 **\*** T(i).vindices[2] **+** 1] **-**              model**->**vertices[3 **\*** T(i).vindices[0] **+** 1];          v[2] **=** model**->**vertices[3 **\*** T(i).vindices[2] **+** 2] **-**              model**->**vertices[3 **\*** T(i).vindices[0] **+** 2];            glmCross(u, v, **&**model**->**facetnorms[3 **\*** (i**+**1)]);          glmNormalize(**&**model**->**facetnorms[3 **\*** (i**+**1)]);      }  }  GLvoid  **glmVertexNormals**(GLMmodel**\*** model, GLfloat angle)  {      GLMnode**\*** node;      GLMnode**\*** tail;      GLMnode**\*\*** members;      GLfloat**\*** normals;      GLuint numnormals;      GLfloat average[3];      GLfloat dot, cos\_angle;      GLuint i, avg;        assert(model);      assert(model**->**facetnorms);        cos\_angle **=** cos(angle **\*** M\_PI **/** 180.0);    **if** (model**->**normals)          free(model**->**normals);        model**->**numnormals **=** model**->**numtriangles **\*** 3;      model**->**normals **=** (GLfloat**\***)malloc(**sizeof**(GLfloat)**\*** 3**\*** (model**->**numnormals**+**1));        members **=** (GLMnode**\*\***)malloc(**sizeof**(GLMnode**\***) **\*** (model**->**numvertices **+** 1));  **for** (i **=** 1; i **<=** model**->**numvertices; i**++**)          members[i] **=** NULL;    **for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {          node **=** (GLMnode**\***)malloc(**sizeof**(GLMnode));          node**->**index **=** i;          node**->**next **=** members[T(i).vindices[0]];          members[T(i).vindices[0]] **=** node;            node **=** (GLMnode**\***)malloc(**sizeof**(GLMnode));          node**->**index **=** i;          node**->**next **=** members[T(i).vindices[1]];          members[T(i).vindices[1]] **=** node;            node **=** (GLMnode**\***)malloc(**sizeof**(GLMnode));          node**->**index **=** i;          node**->**next **=** members[T(i).vindices[2]];          members[T(i).vindices[2]] **=** node;      }        numnormals **=** 1;  **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {          node **=** members[i];  **if** (**!**node)              fprintf(stderr, "glmVertexNormals(): vertex w/o a triangle\n");          average[0] **=** 0.0; average[1] **=** 0.0; average[2] **=** 0.0;          avg **=** 0;  **while** (node) {              dot **=** glmDot(**&**model**->**facetnorms[3 **\*** T(node**->**index).findex],  **&**model**->**facetnorms[3 **\*** T(members[i]**->**index).findex]);  **if** (dot **>** cos\_angle) {                  node**->**averaged **=** GL\_TRUE;                  average[0] **+=** model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 0];                  average[1] **+=** model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 1];                  average[2] **+=** model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 2];                  avg **=** 1; */\* we averaged at least one normal! \*/*              } **else** {                  node**->**averaged **=** GL\_FALSE;              }              node **=** node**->**next;          }    **if** (avg) {              glmNormalize(average);                model**->**normals[3 **\*** numnormals **+** 0] **=** average[0];              model**->**normals[3 **\*** numnormals **+** 1] **=** average[1];              model**->**normals[3 **\*** numnormals **+** 2] **=** average[2];              avg **=** numnormals;              numnormals**++**;          }            node **=** members[i];  **while** (node) {  **if** (node**->**averaged) {  **if** (T(node**->**index).vindices[0] **==** i)                      T(node**->**index).nindices[0] **=** avg;  **else** **if** (T(node**->**index).vindices[1] **==** i)                      T(node**->**index).nindices[1] **=** avg;  **else** **if** (T(node**->**index).vindices[2] **==** i)                      T(node**->**index).nindices[2] **=** avg;              } **else** {                  model**->**normals[3 **\*** numnormals **+** 0] **=**                      model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 0];                  model**->**normals[3 **\*** numnormals **+** 1] **=**                      model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 1];                  model**->**normals[3 **\*** numnormals **+** 2] **=**                      model**->**facetnorms[3 **\*** T(node**->**index).findex **+** 2];  **if** (T(node**->**index).vindices[0] **==** i)                      T(node**->**index).nindices[0] **=** numnormals;  **else** **if** (T(node**->**index).vindices[1] **==** i)                      T(node**->**index).nindices[1] **=** numnormals;  **else** **if** (T(node**->**index).vindices[2] **==** i)                      T(node**->**index).nindices[2] **=** numnormals;                  numnormals**++**;              }              node **=** node**->**next;          }      }        model**->**numnormals **=** numnormals **-** 1;    **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {          node **=** members[i];  **while** (node) {              tail **=** node;              node **=** node**->**next;              free(tail);          }      }      free(members);        normals **=** model**->**normals;      model**->**normals **=** (GLfloat**\***)malloc(**sizeof**(GLfloat)**\*** 3**\*** (model**->**numnormals**+**1));  **for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {          model**->**normals[3 **\*** i **+** 0] **=** normals[3 **\*** i **+** 0];          model**->**normals[3 **\*** i **+** 1] **=** normals[3 **\*** i **+** 1];          model**->**normals[3 **\*** i **+** 2] **=** normals[3 **\*** i **+** 2];      }      free(normals);  }  GLvoid  **glmLinearTexture**(GLMmodel**\*** model)  {      GLMgroup **\***group;      GLfloat dimensions[3];      GLfloat x, y, scalefactor;      GLuint i;        assert(model);    **if** (model**->**texcoords)          free(model**->**texcoords);      model**->**numtexcoords **=** model**->**numvertices;      model**->**texcoords**=**(GLfloat**\***)malloc(**sizeof**(GLfloat)**\***2**\***(model**->**numtexcoords**+**1));        glmDimensions(model, dimensions);      scalefactor **=** 2.0 **/**          glmAbs(glmMax(glmMax(dimensions[0], dimensions[1]), dimensions[2]));    **for**(i **=** 1; i **<=** model**->**numvertices; i**++**) {          x **=** model**->**vertices[3 **\*** i **+** 0] **\*** scalefactor;          y **=** model**->**vertices[3 **\*** i **+** 2] **\*** scalefactor;          model**->**texcoords[2 **\*** i **+** 0] **=** (x **+** 1.0) **/** 2.0;          model**->**texcoords[2 **\*** i **+** 1] **=** (y **+** 1.0) **/** 2.0;      }        group **=** model**->**groups;  **while**(group) {  **for**(i **=** 0; i **<** group**->**numtriangles; i**++**) {              T(group**->**triangles[i]).tindices[0] **=** T(group**->**triangles[i]).vindices[0];              T(group**->**triangles[i]).tindices[1] **=** T(group**->**triangles[i]).vindices[1];              T(group**->**triangles[i]).tindices[2] **=** T(group**->**triangles[i]).vindices[2];          }          group **=** group**->**next;      }    **#if 0**  *printf("glmLinearTexture(): generated %d linear texture coordinates\n",*  *model->numtexcoords);*  **#endif**  }  GLvoid  **glmSpheremapTexture**(GLMmodel**\*** model)  {      GLMgroup**\*** group;      GLfloat theta, phi, rho, x, y, z, r;      GLuint i;        assert(model);      assert(model**->**normals);    **if** (model**->**texcoords)          free(model**->**texcoords);      model**->**numtexcoords **=** model**->**numnormals;      model**->**texcoords**=**(GLfloat**\***)malloc(**sizeof**(GLfloat)**\***2**\***(model**->**numtexcoords**+**1));    **for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {          z **=** model**->**normals[3 **\*** i **+** 0]; */\* re-arrange for pole distortion \*/*          y **=** model**->**normals[3 **\*** i **+** 1];          x **=** model**->**normals[3 **\*** i **+** 2];          r **=** sqrt((x **\*** x) **+** (y **\*** y));          rho **=** sqrt((r **\*** r) **+** (z **\*** z));    **if**(r **==** 0.0) {              theta **=** 0.0;              phi **=** 0.0;          } **else** {  **if**(z **==** 0.0)                  phi **=** 3.14159265 **/** 2.0;  **else**                  phi **=** acos(z **/** rho);    **if**(y **==** 0.0)                  theta **=** 3.141592365 **/** 2.0;  **else**                  theta **=** asin(y **/** r) **+** (3.14159265 **/** 2.0);          }            model**->**texcoords[2 **\*** i **+** 0] **=** theta **/** 3.14159265;          model**->**texcoords[2 **\*** i **+** 1] **=** phi **/** 3.14159265;      }        group **=** model**->**groups;  **while**(group) {  **for** (i **=** 0; i **<** group**->**numtriangles; i**++**) {              T(group**->**triangles[i]).tindices[0] **=** T(group**->**triangles[i]).nindices[0];              T(group**->**triangles[i]).tindices[1] **=** T(group**->**triangles[i]).nindices[1];              T(group**->**triangles[i]).tindices[2] **=** T(group**->**triangles[i]).nindices[2];          }          group **=** group**->**next;      }  }  GLvoid  **glmDelete**(GLMmodel**\*** model)  {      GLMgroup**\*** group;      GLuint i;        assert(model);    **if** (model**->**pathname) free(model**->**pathname);  **if** (model**->**mtllibname) free(model**->**mtllibname);  **if** (model**->**vertices) free(model**->**vertices);  **if** (model**->**normals) free(model**->**normals);  **if** (model**->**texcoords) free(model**->**texcoords);  **if** (model**->**facetnorms) free(model**->**facetnorms);  **if** (model**->**triangles) free(model**->**triangles);  **if** (model**->**materials) {  **for** (i **=** 0; i **<** model**->**nummaterials; i**++**)              free(model**->**materials[i].name);      }      free(model**->**materials);  **while**(model**->**groups) {          group **=** model**->**groups;          model**->**groups **=** model**->**groups**->**next;          free(group**->**name);          free(group**->**triangles);          free(group);      }        free(model);  }  GLMmodel**\***  **glmReadOBJ**(**char\*** filename)  {      GLMmodel**\*** model;  **FILE\*** file;        file **=** fopen(filename, "r");  **if** (**!**file) {          fprintf(stderr, "glmReadOBJ() failed: can't open data file \"%s\".\n",              filename);          exit(1);      }        model **=** (GLMmodel**\***)malloc(**sizeof**(GLMmodel));      model**->**pathname **=** strdup(filename);      model**->**mtllibname **=** NULL;      model**->**numvertices **=** 0;      model**->**vertices **=** NULL;      model**->**numnormals **=** 0;      model**->**normals **=** NULL;      model**->**numtexcoords **=** 0;      model**->**texcoords **=** NULL;      model**->**numfacetnorms **=** 0;      model**->**facetnorms **=** NULL;      model**->**numtriangles **=** 0;      model**->**triangles **=** NULL;      model**->**nummaterials **=** 0;      model**->**materials **=** NULL;      model**->**numgroups **=** 0;      model**->**groups **=** NULL;      model**->**position[0] **=** 0.0;      model**->**position[1] **=** 0.0;      model**->**position[2] **=** 0.0;        glmFirstPass(model, file);        model**->**vertices **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***          3 **\*** (model**->**numvertices **+** 1));      model**->**triangles **=** (GLMtriangle**\***)malloc(**sizeof**(GLMtriangle) **\***          model**->**numtriangles);  **if** (model**->**numnormals) {          model**->**normals **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***              3 **\*** (model**->**numnormals **+** 1));      }  **if** (model**->**numtexcoords) {          model**->**texcoords **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***              2 **\*** (model**->**numtexcoords **+** 1));      }        rewind(file);        glmSecondPass(model, file);        fclose(file);    **return** model;  }  GLvoid  **glmWriteOBJ**(GLMmodel**\*** model, **char\*** filename, GLuint mode)  {      GLuint i;  **FILE\*** file;      GLMgroup**\*** group;        assert(model);    **if** (mode **&** GLM\_FLAT **&&** **!**model**->**facetnorms) {          printf("glmWriteOBJ() warning: flat normal output requested "              "with no facet normals defined.\n");          mode **&=** **~**GLM\_FLAT;      }  **if** (mode **&** GLM\_SMOOTH **&&** **!**model**->**normals) {          printf("glmWriteOBJ() warning: smooth normal output requested "              "with no normals defined.\n");          mode **&=** **~**GLM\_SMOOTH;      }  **if** (mode **&** GLM\_TEXTURE **&&** **!**model**->**texcoords) {          printf("glmWriteOBJ() warning: texture coordinate output requested "              "with no texture coordinates defined.\n");          mode **&=** **~**GLM\_TEXTURE;      }  **if** (mode **&** GLM\_FLAT **&&** mode **&** GLM\_SMOOTH) {          printf("glmWriteOBJ() warning: flat normal output requested "              "and smooth normal output requested (using smooth).\n");          mode **&=** **~**GLM\_FLAT;      }  **if** (mode **&** GLM\_COLOR **&&** **!**model**->**materials) {          printf("glmWriteOBJ() warning: color output requested "              "with no colors (materials) defined.\n");          mode **&=** **~**GLM\_COLOR;      }  **if** (mode **&** GLM\_MATERIAL **&&** **!**model**->**materials) {          printf("glmWriteOBJ() warning: material output requested "              "with no materials defined.\n");          mode **&=** **~**GLM\_MATERIAL;      }  **if** (mode **&** GLM\_COLOR **&&** mode **&** GLM\_MATERIAL) {          printf("glmWriteOBJ() warning: color and material output requested "              "outputting only materials.\n");          mode **&=** **~**GLM\_COLOR;      }          file **=** fopen(filename, "w");  **if** (**!**file) {          fprintf(stderr, "glmWriteOBJ() failed: can't open file \"%s\" to write.\n",              filename);          exit(1);      }        fprintf(file, "# \n");      fprintf(file, "# Wavefront OBJ generated by GLM library\n");      fprintf(file, "# \n");      fprintf(file, "# GLM library\n");      fprintf(file, "# Nate Robins\n");      fprintf(file, "# ndr@pobox.com\n");      fprintf(file, "# http://www.pobox.com/~ndr\n");      fprintf(file, "# \n");    **if** (mode **&** GLM\_MATERIAL **&&** model**->**mtllibname) {          fprintf(file, "\nmtllib %s\n\n", model**->**mtllibname);          glmWriteMTL(model, filename, model**->**mtllibname);      }        fprintf(file, "\n");      fprintf(file, "# %d vertices\n", model**->**numvertices);  **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {          fprintf(file, "v %f %f %f\n",              model**->**vertices[3 **\*** i **+** 0],              model**->**vertices[3 **\*** i **+** 1],              model**->**vertices[3 **\*** i **+** 2]);      }    **if** (mode **&** GLM\_SMOOTH) {          fprintf(file, "\n");          fprintf(file, "# %d normals\n", model**->**numnormals);  **for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {              fprintf(file, "vn %f %f %f\n",                  model**->**normals[3 **\*** i **+** 0],                  model**->**normals[3 **\*** i **+** 1],                  model**->**normals[3 **\*** i **+** 2]);          }      } **else** **if** (mode **&** GLM\_FLAT) {          fprintf(file, "\n");          fprintf(file, "# %d normals\n", model**->**numfacetnorms);  **for** (i **=** 1; i **<=** model**->**numnormals; i**++**) {              fprintf(file, "vn %f %f %f\n",                  model**->**facetnorms[3 **\*** i **+** 0],                  model**->**facetnorms[3 **\*** i **+** 1],                  model**->**facetnorms[3 **\*** i **+** 2]);          }      }    **if** (mode **&** GLM\_TEXTURE) {          fprintf(file, "\n");          fprintf(file, "# %d texcoords\n", model**->**numtexcoords);  **for** (i **=** 1; i **<=** model**->**numtexcoords; i**++**) {              fprintf(file, "vt %f %f\n",                  model**->**texcoords[2 **\*** i **+** 0],                  model**->**texcoords[2 **\*** i **+** 1]);          }      }        fprintf(file, "\n");      fprintf(file, "# %d groups\n", model**->**numgroups);      fprintf(file, "# %d faces (triangles)\n", model**->**numtriangles);      fprintf(file, "\n");        group **=** model**->**groups;  **while**(group) {          fprintf(file, "g %s\n", group**->**name);  **if** (mode **&** GLM\_MATERIAL)              fprintf(file, "usemtl %s\n", model**->**materials[group**->**material].name);  **for** (i **=** 0; i **<** group**->**numtriangles; i**++**) {  **if** (mode **&** GLM\_SMOOTH **&&** mode **&** GLM\_TEXTURE) {                  fprintf(file, "f %d/%d/%d %d/%d/%d %d/%d/%d\n",                      T(group**->**triangles[i]).vindices[0],                      T(group**->**triangles[i]).tindices[0],                      T(group**->**triangles[i]).nindices[0],                      T(group**->**triangles[i]).vindices[1],                      T(group**->**triangles[i]).tindices[1],                      T(group**->**triangles[i]).nindices[1],                      T(group**->**triangles[i]).vindices[2],                      T(group**->**triangles[i]).tindices[2],                      T(group**->**triangles[i]).nindices[2]);              } **else** **if** (mode **&** GLM\_FLAT **&&** mode **&** GLM\_TEXTURE) {                  fprintf(file, "f %d/%d %d/%d %d/%d\n",                      T(group**->**triangles[i]).vindices[0],                      T(group**->**triangles[i]).findex,                      T(group**->**triangles[i]).vindices[1],                      T(group**->**triangles[i]).findex,                      T(group**->**triangles[i]).vindices[2],                      T(group**->**triangles[i]).findex);              } **else** **if** (mode **&** GLM\_TEXTURE) {                  fprintf(file, "f %d/%d %d/%d %d/%d\n",                      T(group**->**triangles[i]).vindices[0],                      T(group**->**triangles[i]).tindices[0],                      T(group**->**triangles[i]).vindices[1],                      T(group**->**triangles[i]).tindices[1],                      T(group**->**triangles[i]).vindices[2],                      T(group**->**triangles[i]).tindices[2]);              } **else** **if** (mode **&** GLM\_SMOOTH) {                  fprintf(file, "f %d//%d %d//%d %d//%d\n",                      T(group**->**triangles[i]).vindices[0],                      T(group**->**triangles[i]).nindices[0],                      T(group**->**triangles[i]).vindices[1],                      T(group**->**triangles[i]).nindices[1],                      T(group**->**triangles[i]).vindices[2],                      T(group**->**triangles[i]).nindices[2]);              } **else** **if** (mode **&** GLM\_FLAT) {                  fprintf(file, "f %d//%d %d//%d %d//%d\n",                      T(group**->**triangles[i]).vindices[0],                      T(group**->**triangles[i]).findex,                      T(group**->**triangles[i]).vindices[1],                      T(group**->**triangles[i]).findex,                      T(group**->**triangles[i]).vindices[2],                      T(group**->**triangles[i]).findex);              } **else** {                  fprintf(file, "f %d %d %d\n",                      T(group**->**triangles[i]).vindices[0],                      T(group**->**triangles[i]).vindices[1],                      T(group**->**triangles[i]).vindices[2]);              }          }          fprintf(file, "\n");          group **=** group**->**next;      }        fclose(file);  }  GLvoid  **glmDraw**(GLMmodel**\*** model, GLuint mode)  {  **static** GLuint i;  **static** GLMgroup**\*** group;  **static** GLMtriangle**\*** triangle;  **static** GLMmaterial**\*** material;        assert(model);      assert(model**->**vertices);    **if** (mode **&** GLM\_FLAT **&&** **!**model**->**facetnorms) {          printf("glmDraw() warning: flat render mode requested "              "with no facet normals defined.\n");          mode **&=** **~**GLM\_FLAT;      }  **if** (mode **&** GLM\_SMOOTH **&&** **!**model**->**normals) {          printf("glmDraw() warning: smooth render mode requested "              "with no normals defined.\n");          mode **&=** **~**GLM\_SMOOTH;      }  **if** (mode **&** GLM\_TEXTURE **&&** **!**model**->**texcoords) {          printf("glmDraw() warning: texture render mode requested "              "with no texture coordinates defined.\n");          mode **&=** **~**GLM\_TEXTURE;      }  **if** (mode **&** GLM\_FLAT **&&** mode **&** GLM\_SMOOTH) {          printf("glmDraw() warning: flat render mode requested "              "and smooth render mode requested (using smooth).\n");          mode **&=** **~**GLM\_FLAT;      }  **if** (mode **&** GLM\_COLOR **&&** **!**model**->**materials) {          printf("glmDraw() warning: color render mode requested "              "with no materials defined.\n");          mode **&=** **~**GLM\_COLOR;      }  **if** (mode **&** GLM\_MATERIAL **&&** **!**model**->**materials) {          printf("glmDraw() warning: material render mode requested "              "with no materials defined.\n");          mode **&=** **~**GLM\_MATERIAL;      }  **if** (mode **&** GLM\_COLOR **&&** mode **&** GLM\_MATERIAL) {          printf("glmDraw() warning: color and material render mode requested "              "using only material mode.\n");          mode **&=** **~**GLM\_COLOR;      }  **if** (mode **&** GLM\_COLOR)          glEnable(GL\_COLOR\_MATERIAL);  **else** **if** (mode **&** GLM\_MATERIAL)          glDisable(GL\_COLOR\_MATERIAL);          group **=** model**->**groups;  **while** (group) {  **if** (mode **&** GLM\_MATERIAL) {              material **=** **&**model**->**materials[group**->**material];              glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, material**->**ambient);              glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, material**->**diffuse);              glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, material**->**specular);              glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, material**->**shininess);          }    **if** (mode **&** GLM\_COLOR) {              glColor3fv(material**->**diffuse);          }            glBegin(GL\_TRIANGLES);  **for** (i **=** 0; i **<** group**->**numtriangles; i**++**) {              triangle **=** **&**T(group**->**triangles[i]);    **if** (mode **&** GLM\_FLAT)                  glNormal3fv(**&**model**->**facetnorms[3 **\*** triangle**->**findex]);    **if** (mode **&** GLM\_SMOOTH)                  glNormal3fv(**&**model**->**normals[3 **\*** triangle**->**nindices[0]]);  **if** (mode **&** GLM\_TEXTURE)                  glTexCoord2fv(**&**model**->**texcoords[2 **\*** triangle**->**tindices[0]]);              glVertex3fv(**&**model**->**vertices[3 **\*** triangle**->**vindices[0]]);    **if** (mode **&** GLM\_SMOOTH)                  glNormal3fv(**&**model**->**normals[3 **\*** triangle**->**nindices[1]]);  **if** (mode **&** GLM\_TEXTURE)                  glTexCoord2fv(**&**model**->**texcoords[2 **\*** triangle**->**tindices[1]]);              glVertex3fv(**&**model**->**vertices[3 **\*** triangle**->**vindices[1]]);    **if** (mode **&** GLM\_SMOOTH)                  glNormal3fv(**&**model**->**normals[3 **\*** triangle**->**nindices[2]]);  **if** (mode **&** GLM\_TEXTURE)                  glTexCoord2fv(**&**model**->**texcoords[2 **\*** triangle**->**tindices[2]]);              glVertex3fv(**&**model**->**vertices[3 **\*** triangle**->**vindices[2]]);            }          glEnd();            group **=** group**->**next;      }  }  GLuint  **glmList**(GLMmodel**\*** model, GLuint mode)  {      GLuint list;        list **=** glGenLists(1);      glNewList(list, GL\_COMPILE);      glmDraw(model, mode);      glEndList();    **return** list;  }  GLvoid  **glmWeld**(GLMmodel**\*** model, GLfloat epsilon)  {      GLfloat**\*** vectors;      GLfloat**\*** copies;      GLuint numvectors;      GLuint i;        numvectors **=** model**->**numvertices;      vectors **=** model**->**vertices;      copies **=** glmWeldVectors(vectors, **&**numvectors, epsilon);    **#if 0**  *printf("glmWeld(): %d redundant vertices.\n",*  *model->numvertices - numvectors - 1);*  **#endif**    **for** (i **=** 0; i **<** model**->**numtriangles; i**++**) {          T(i).vindices[0] **=** (GLuint)vectors[3 **\*** T(i).vindices[0] **+** 0];          T(i).vindices[1] **=** (GLuint)vectors[3 **\*** T(i).vindices[1] **+** 0];          T(i).vindices[2] **=** (GLuint)vectors[3 **\*** T(i).vindices[2] **+** 0];      }        free(vectors);        model**->**numvertices **=** numvectors;      model**->**vertices **=** (GLfloat**\***)malloc(**sizeof**(GLfloat) **\***          3 **\*** (model**->**numvertices **+** 1));    **for** (i **=** 1; i **<=** model**->**numvertices; i**++**) {          model**->**vertices[3 **\*** i **+** 0] **=** copies[3 **\*** i **+** 0];          model**->**vertices[3 **\*** i **+** 1] **=** copies[3 **\*** i **+** 1];          model**->**vertices[3 **\*** i **+** 2] **=** copies[3 **\*** i **+** 2];      }        free(copies);  }  GLubyte**\***  **glmReadPPM**(**char\*** filename, **int\*** width, **int\*** height)  {  **FILE\*** fp;  **int** i, w, h, d;  **unsigned** **char\*** image;  **char** head[70];        fp **=** fopen(filename, "rb");  **if** (**!**fp) {          perror(filename);  **return** NULL;      }        fgets(head, 70, fp);  **if** (strncmp(head, "P6", 2)) {          fprintf(stderr, "%s: Not a raw PPM file\n", filename);  **return** NULL;      }        i **=** 0;  **while**(i **<** 3) {          fgets(head, 70, fp);  **if** (head[0] **==** '#')  **continue**;  **if** (i **==** 0)              i **+=** sscanf(head, "%d %d %d", **&**w, **&**h, **&**d);  **else** **if** (i **==** 1)              i **+=** sscanf(head, "%d %d", **&**h, **&**d);  **else** **if** (i **==** 2)              i **+=** sscanf(head, "%d", **&**d);      }        image **=** (**unsigned** **char\***)malloc(**sizeof**(**unsigned** **char**)**\***w**\***h**\***3);      fread(image, **sizeof**(**unsigned** **char**), w**\***h**\***3, fp);      fclose(fp);    **\***width **=** w;  **\***height **=** h;  **return** image;  }  **#if 0**  *if (model->numnormals) {*  *numvectors = model->numnormals;*  *vectors = model->normals;*  *copies = glmOptimizeVectors(vectors, &numvectors);*    *printf("glmOptimize(): %d redundant normals.\n",*  *model->numnormals - numvectors);*    *for (i = 0; i < model->numtriangles; i++) {*  *T(i).nindices[0] = (GLuint)vectors[3 \* T(i).nindices[0] + 0];*  *T(i).nindices[1] = (GLuint)vectors[3 \* T(i).nindices[1] + 0];*  *T(i).nindices[2] = (GLuint)vectors[3 \* T(i).nindices[2] + 0];*  *}*    *model->numnormals = numvectors;*  *model->normals = (GLfloat\*)malloc(sizeof(GLfloat) \**  *3 \* (model->numnormals + 1));*    *for (i = 1; i <= model->numnormals; i++) {*  *model->normals[3 \* i + 0] = copies[3 \* i + 0];*  *model->normals[3 \* i + 1] = copies[3 \* i + 1];*  *model->normals[3 \* i + 2] = copies[3 \* i + 2];*  *}*    *free(copies);*  *}*  *if (model->numtexcoords) {*  *numvectors = model->numtexcoords;*  *vectors = model->texcoords;*  *copies = glmOptimizeVectors(vectors, &numvectors);*    *printf("glmOptimize(): %d redundant texcoords.\n",*  *model->numtexcoords - numvectors);*    *for (i = 0; i < model->numtriangles; i++) {*  *for (j = 0; j < 3; j++) {*  *T(i).tindices[j] = (GLuint)vectors[3 \* T(i).tindices[j] + 0];*  *}*  *}*      *free(vectors);*    *model->numtexcoords = numvectors;*  *model->texcoords = (GLfloat\*)malloc(sizeof(GLfloat) \**  *2 \* (model->numtexcoords + 1));*      *for (i = 1; i <= model->numtexcoords; i++) {*  *model->texcoords[2 \* i + 0] = copies[2 \* i + 0];*  *model->texcoords[2 \* i + 1] = copies[2 \* i + 1];*  *}*    *free(copies);*  *}*  **#endif**  **#if 0**  *for (i = 1; i <= model->numvertices; i++) {*  *for (j = 0; j < model->numtriangles; i++) {*  *if (T(j).vindices[0] == i ||*  *T(j).vindices[1] == i ||*  *T(j).vindices[1] == i)*  *break;*  *}*  *}*  **#endif** |

Setelah dicompile dan run, maka hasilnya seperti screenshoot dibawah ini :

|  |
| --- |
|  |
|  |