Mesh 网格模型的切片

项目报告

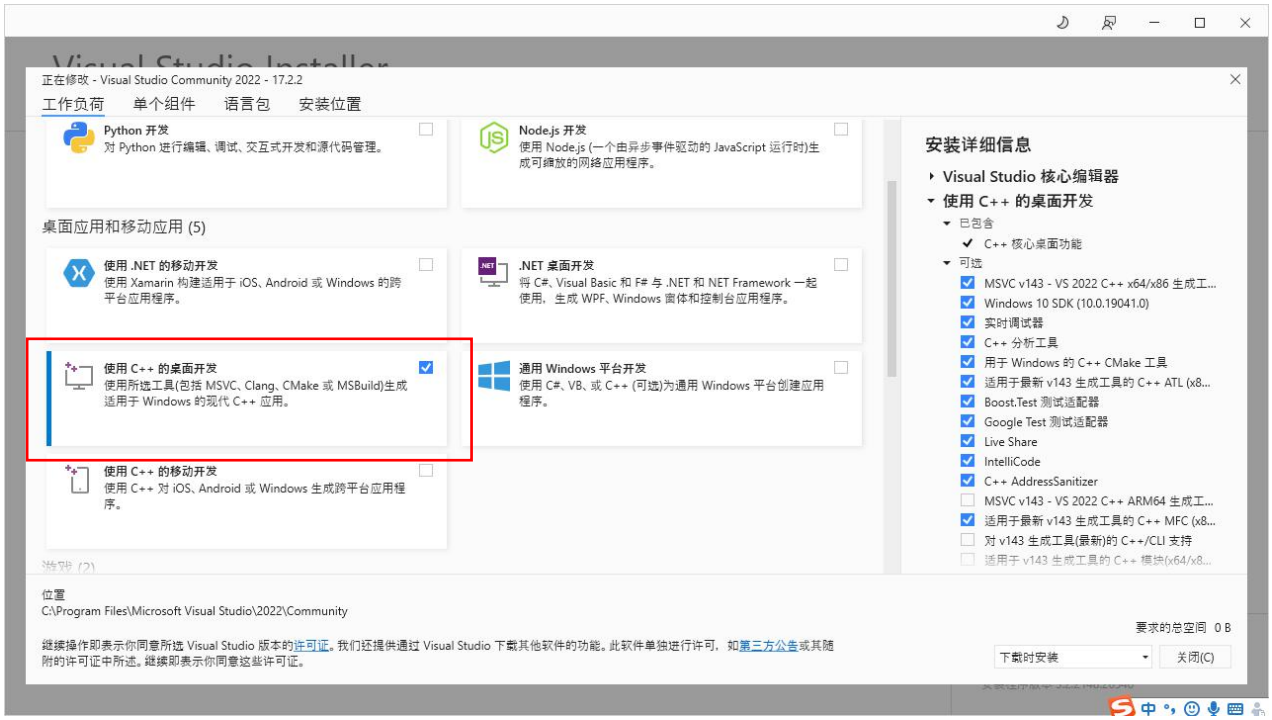
报告人：朱晢宇

目录

1. 所需软件及创建项目

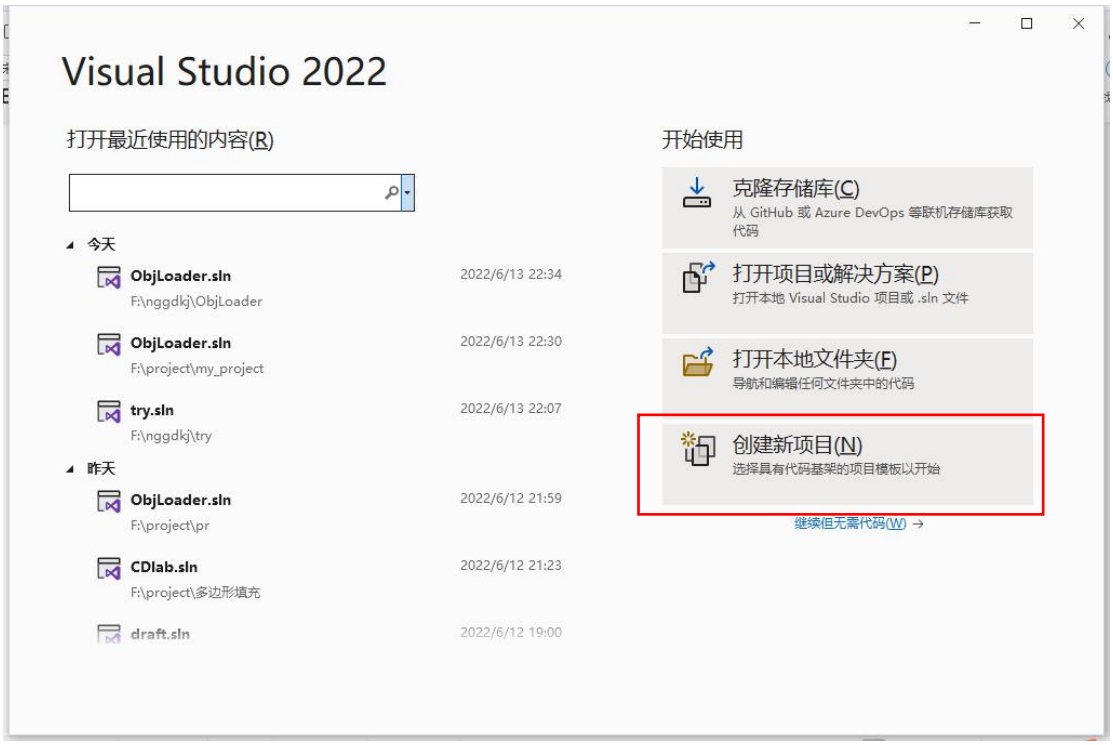
所需软件

安装 Visual Studio 2022；
安装“使用 C++的桌面开发”。



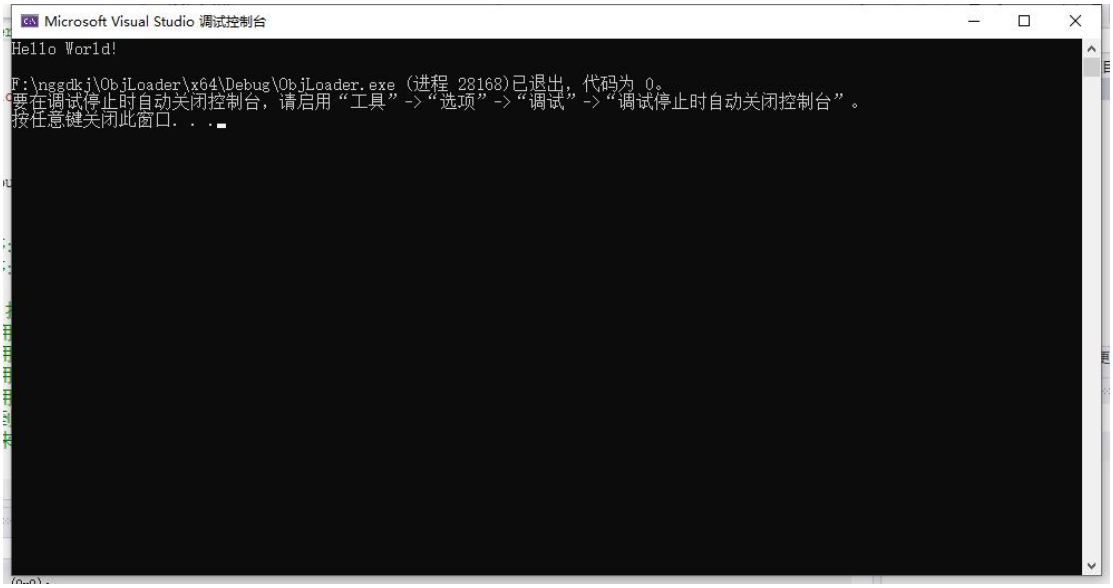
创建项目

点击“创建新项目”；
选择“控制台应用”；
将项目命名为“ObjLoader”，点击创建。


Visual Studio 2022
打开最近使用的内容(R)
今天
ObjLoader.sln
F:\nggdkj\ObjLoader
2022/6/13 22:34
ObjLoader.sln
F:\project\my_project
2022/6/13 22:30
try.sln
F:\nggdkj\try
2022/6/13 22:07
昨天
ObjLoader.sln
F:\project\pr
2022/6/12 21:59
CDlab.sln
F:\project\多边形填充
2022/6/12 21:23
draft.sln
2022/6/12 19:00
开始使用
克隆存储库(C)
从 GitHub 或 Azure DevOps 等联机存储库获取代码
打开项目或解决方案(P)
打开本地 Visual Studio 项目或 .sln 文件
打开本地文件夹(F)
导航和编辑任何文件夹中的代码
创建新项目(N)
选择具有代码基架的项目模板以开始
继续但无需代码(W) →

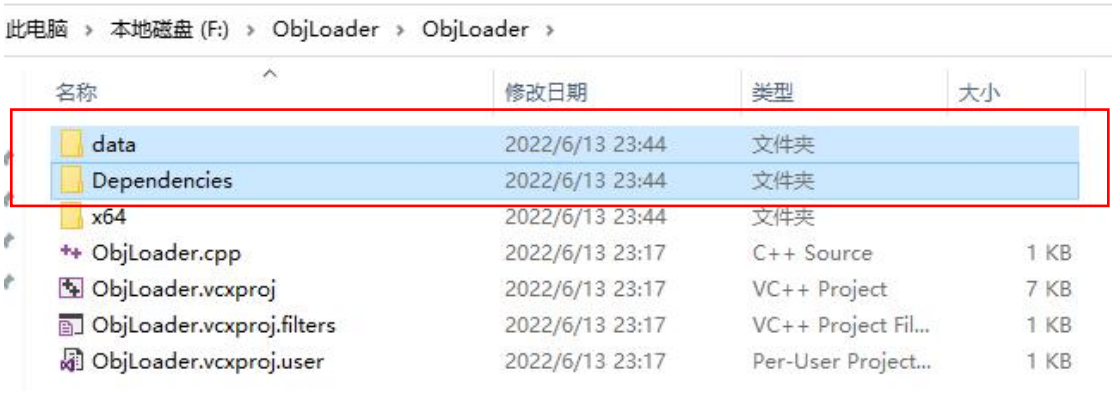

创建新项目
最近使用的项目模板(R)
控制台应用 C++
空项目 C++
MFC 应用 C++
搜索模板(Alt+S)(S)
所有语言(L)
所有平台(P)
所有项目类型(T)
空项目
使用适用于 Windows 的 C++ 从头开始操作。不提供基础文件。
C++ Windows 控制台
控制台应用
在 Windows 终端运行代码。默认打印 "Hello World"。
C++ Windows 控制台
CMake 项目
生成不依赖于 .sln 或 .vcxproj 文件的新式跨平台 C++ 应用。
C++ Windows Linux 控制台
Windows 桌面向导
使用向导自行创建 Windows 应用。
C++ Windows 桌面 控制台 库
Windows 桌面应用程序
具有在 Windows 上运行的图形用户界面的应用程序的项目。
C++ Windows 桌面
上一步(B)
下一步(N)

运行项目，输出“Hello World”，则项目创建成功。

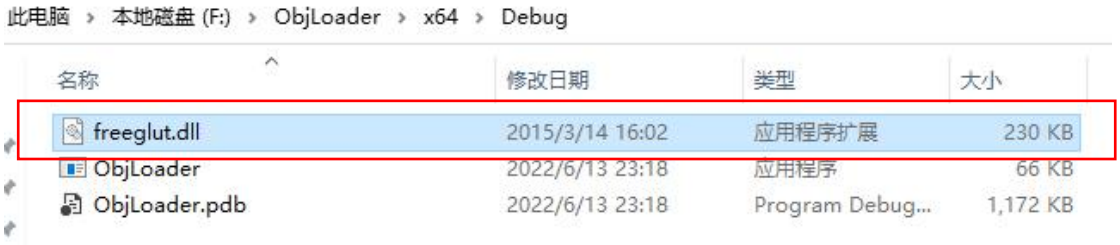


2. 环境配置

（1）复制“环境配置”文件夹中的“data”文件夹和“Dependencies”文件夹，粘贴到项目文件夹中的“ObjLoader”文件夹中。



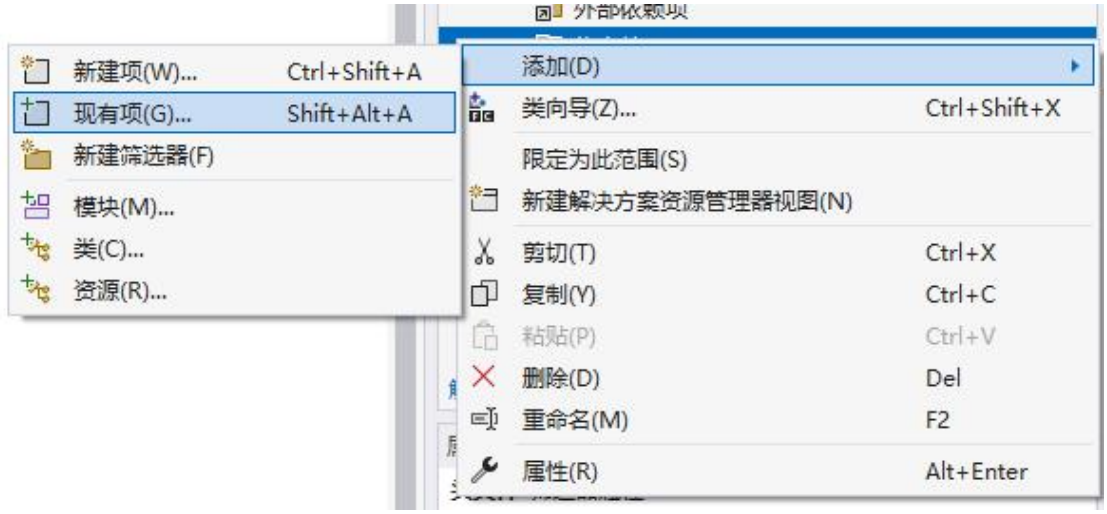
（2）复制“环境配置”文件夹中的“freeglut.dll”，粘贴到项目文件夹中的“x64\Debug”中。



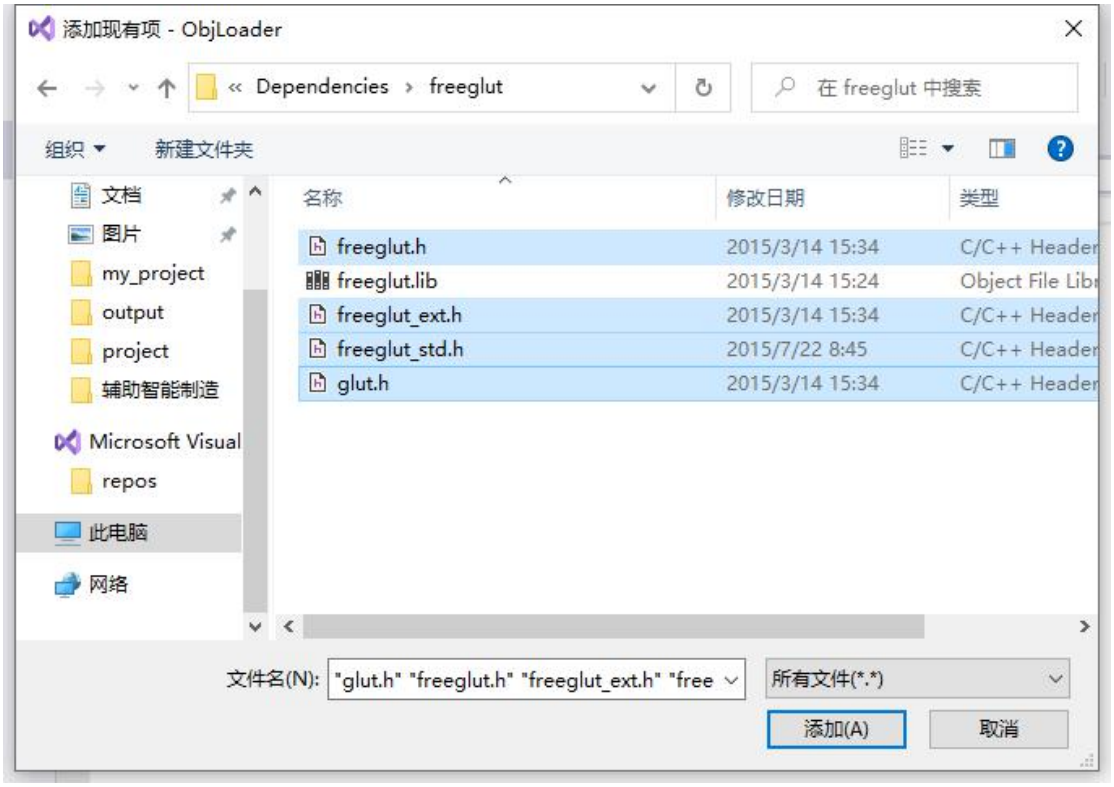
3. 在项目中导入和添加文件

头文件

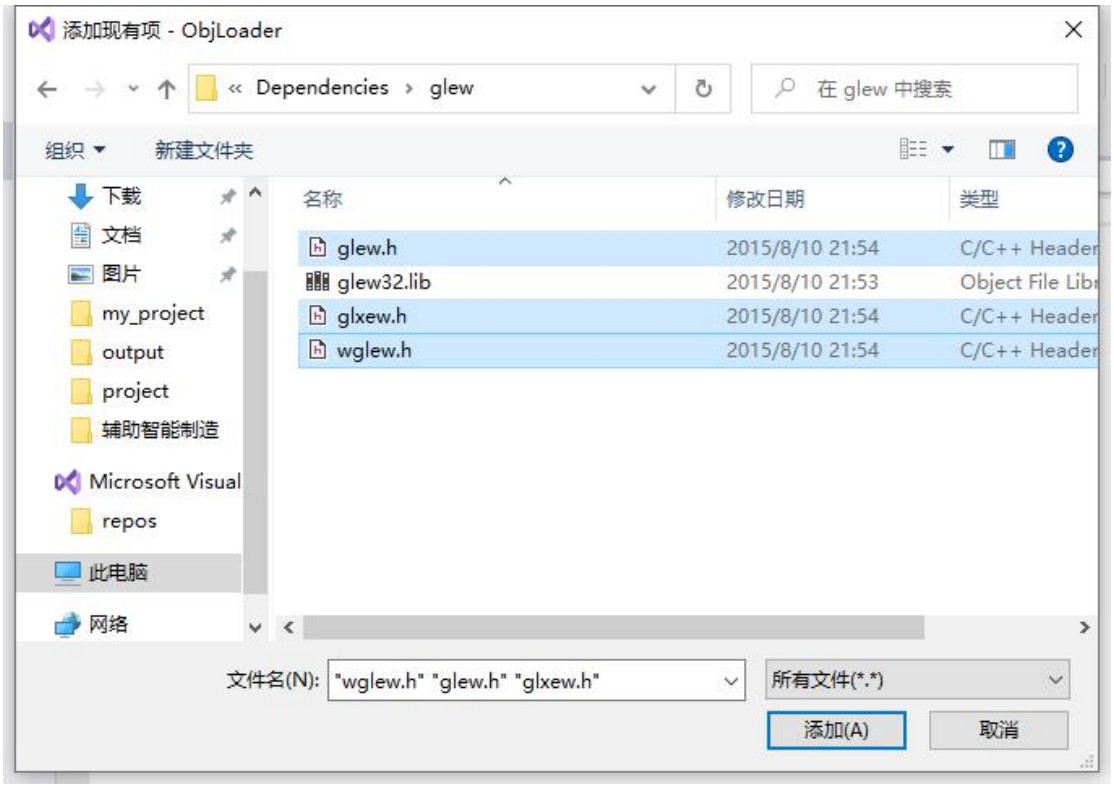
① 在“解决方案资源管理器”中，右击“头文件”，选择“添加—现有项”。



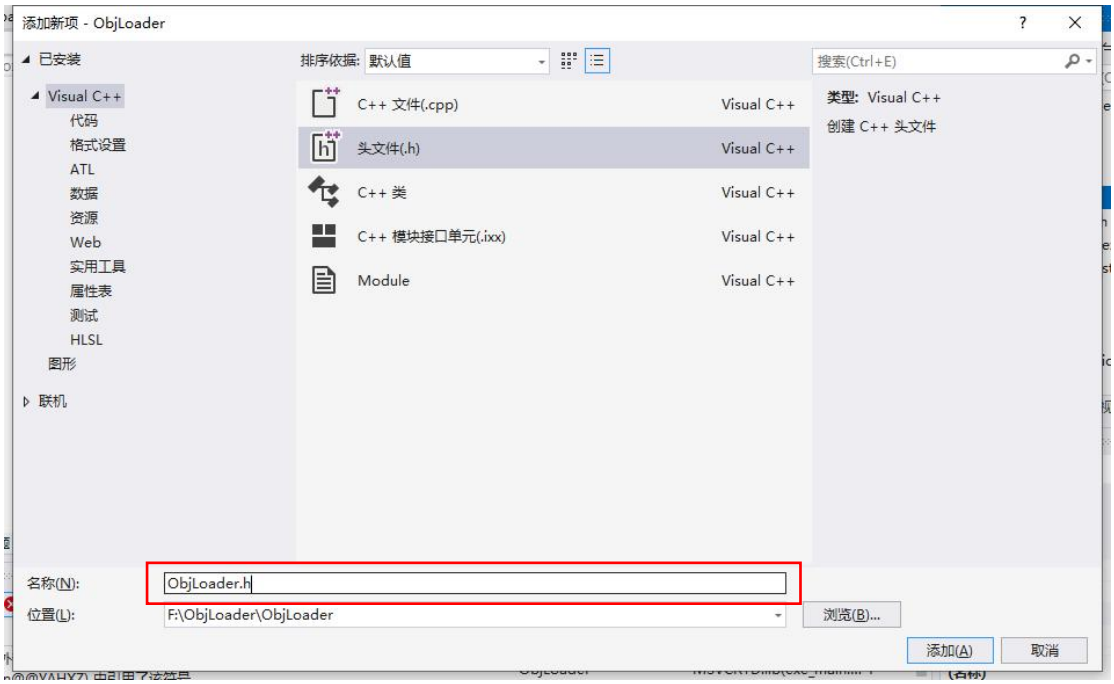
② 添加刚刚的“Dependencies”文件夹下的“freeglut”文件夹中的“freeglut.h”、“freeglut_ext.h”、“freeglut_std.h”、“glut.h”四个头文件。

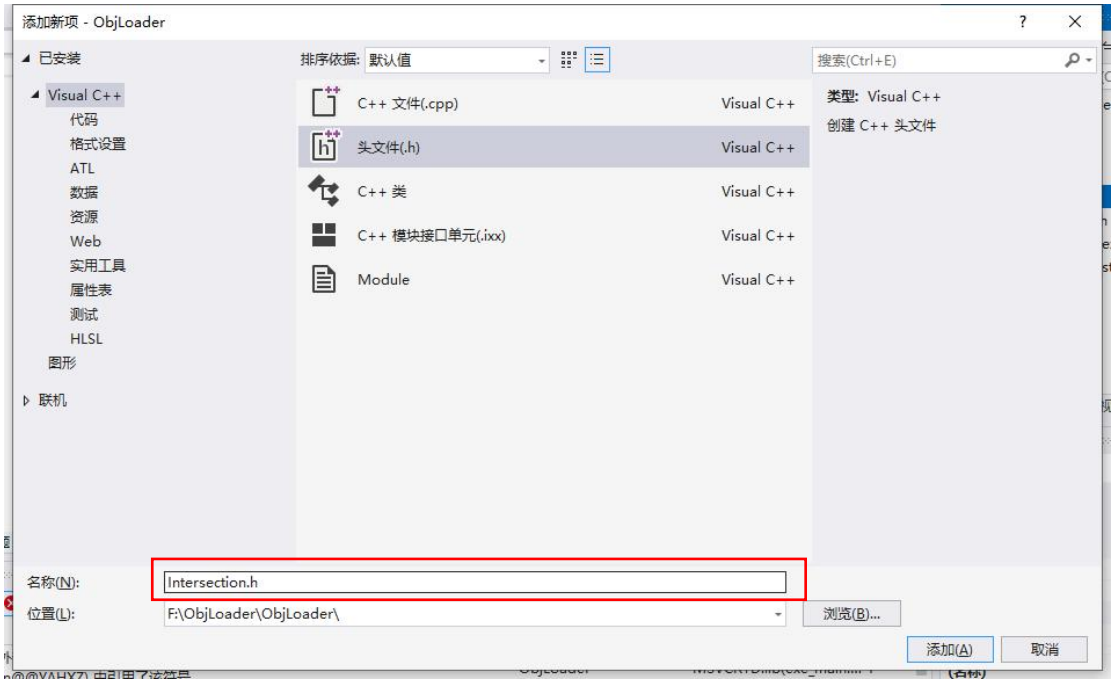


③ 同理，添加“Dependencies”文件夹下的“glew”文件夹中的“glew.h”、“glxew.h”、“wglew.h”三个头文件。



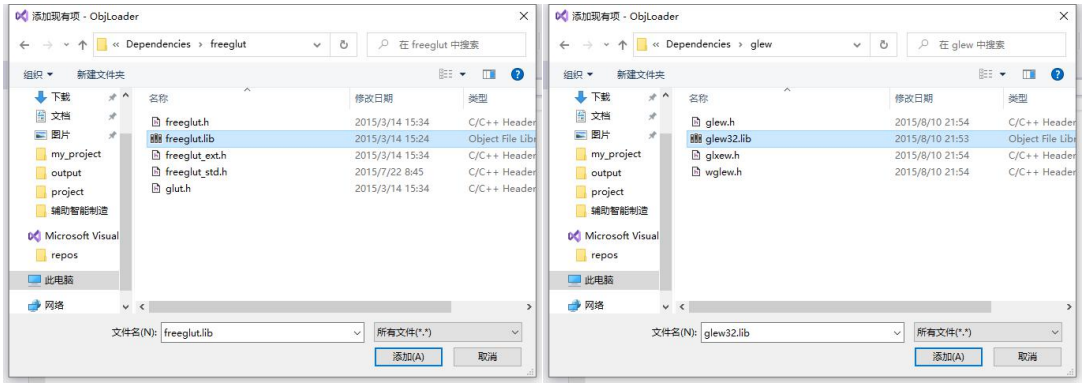
④ 在“解决方案资源管理器”中，右击“头文件”，选择“添加—新建项”。分别新建名称为“ObjLoader.h”和“Intersection.h”的头文件。





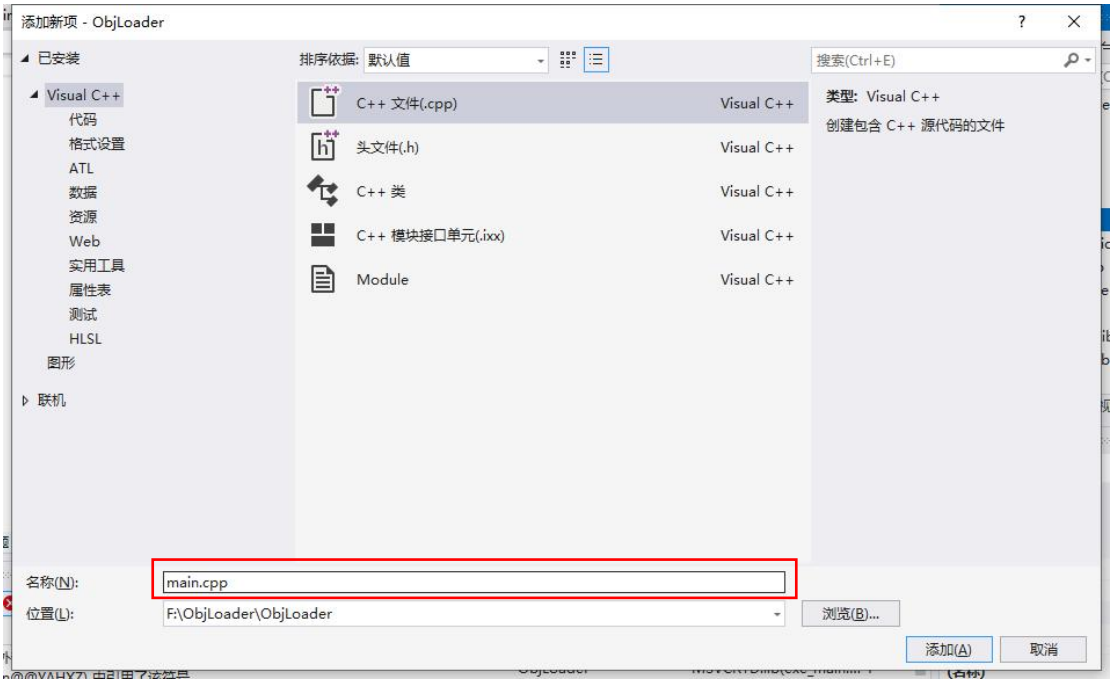
资源文件

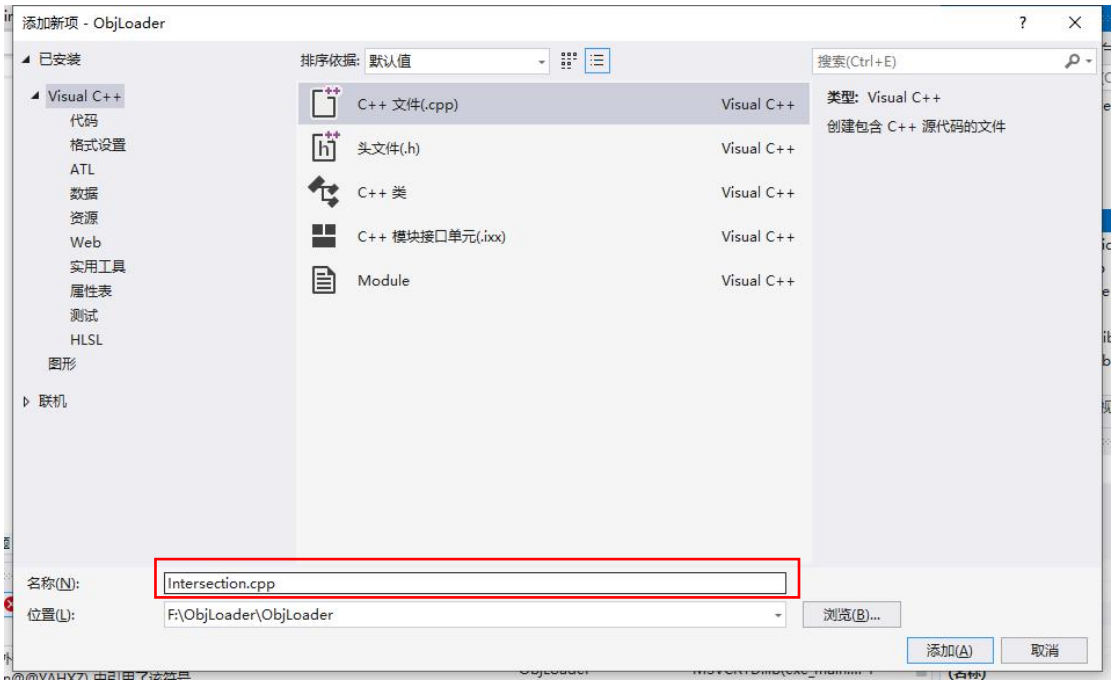
在“解决方案资源管理器”中，右击“资源文件”，选择“添加—现有项”。
添加“freeglut”文件夹中“freeglut.lib”和“glew”文件夹中的“glew32.lib”。



源文件

在“解决方案资源管理器”中，右击“源文件”，选择“添加—新建项”。分别新建名称为“main.cpp”和“Intersection.cpp”的源文件。





4. 编写代码

将以下代码分别粘贴到对应的文件中。

ObjLoader.h

```
#pragma once
#include "Dependencies\glew\glew.h"
#include "Dependencies\freeglut\freeglut.h"
#include "Intersection.h"
#include <vector>
#include <string>
using namespace std;

class ObjLoader {
public:
    ObjLoader(string filename);
    void Draw();
    void slice(int num_of_layers, int layer);
    vector<vector<GLfloat>>vSets;
    vector<vector<GLint>>fSets;
    bool t;
};
```

Intersection.h

```
#pragma once
#include <iostream>
#include <vector>
#include <math.h>
#include "Dependencies\glew\glew.h"
#include "Dependencies\freeglut\freeglut.h"
using namespace std;

class Intersection
{
public:
    vector<GLfloat> v0;
    vector<GLfloat> v1;
    vector<GLfloat> v2;
    Intersection(vector<GLfloat> v_0, vector<GLfloat> v_1, vector<GLfloat> v_2);
};

bool pan(vector<GLfloat> start, vector<GLfloat> end, Intersection tri, vector<GLfloat>&
intersection);
```

ObjLoader.cpp

清空 ObjLoader.cpp 中原先创建项目时自动生成的代码，将以下代码粘贴进去。

```
#include "ObjLoader.h"
#include <fstream>
#include <iostream>
using namespace std;

float x_max = -10000;
float x_min = 10000;

float y_max = -10000;
float y_min = 10000;

float z_max = -10000;
float z_min = 10000;

ObjLoader::ObjLoader(string filename)  //读入 obj 文件
{
    string line;
```

```
fstream f;
t = false;
f.open(filename, ios::in);
while (!f.eof()) {
    getline(f, line);
    vector<string>parameters;
    string tailMark = " ";
    string ans = "";
    line = line.append(tailMark);
    for (int i = 0; i < line.length(); i++) {
        char ch = line[i];
        if (ch != ' ') {
            ans += ch;
        }
        else {
            parameters.push_back(ans);
            ans = "";
        }
    }
    if (parameters[0] == "v")  //读取点
    {
        vector<GLfloat>Point;
        for (int i = 1; i < 4; i++) {
            GLfloat xyz = atof(parameters[i].c_str());
            Point.push_back(xyz);
            //求出该模型的 x_max, x_min, y_max, y_min, z_max, z_min，用于后续切割平
面的设置
            if (i == 1)
            {
                if (x_max <= xyz) x_max = xyz;
                else if (x_min >= xyz) x_min = xyz;
            }
            else if (i == 2)
            {
                if (y_max <= xyz) y_max = xyz;
                else if (y_min >= xyz) y_min = xyz;
            }
            else if (i == 3)
            {
                if (z_max <= xyz) z_max = xyz;
                else if (z_min >= xyz) z_min = xyz;
            }
        }
        vSets.push_back(Point);  //点集
```

```
        }
        else if (parameters[0] == "f") //读取面
        {
            vector<GLint>vIndexSets;
            for (int i = 1; i < 4; i++) {
                string x = parameters[i];
                string ans = "";
                for (int j = 0; j < x.length(); j++) {
                    char ch = x[j];
                    if (ch != '/') {
                        ans += ch;
                    }
                    else {
                        break;
                    }
                }
                GLint index = atof(ans.c_str());  //索引
                index = index--;
                vIndexSets.push_back(index);
            }
            fSets.push_back(vIndexSets);  //面集
        }
    }
    f.close();
}

void ObjLoader::Draw()  //绘制
{
    glBegin(GL_TRIANGLES);
    for (int i = 0; i < fSets.size(); i++) {

        GLfloat VN[3];
        GLfloat SV1[3];
        GLfloat  SV2[3];
        GLfloat SV3[3];

        //三点绘制一个三角面片
        GLint f_v_index = (fSets[i])[0];
        GLint s_v_index = (fSets[i])[1];
        GLint t_v_index = (fSets[i])[2];
        SV1[0] = (vSets[f_v_index])[0];
        SV1[1] = (vSets[f_v_index])[1];
        SV1[2] = (vSets[f_v_index])[2];
        SV2[0] = (vSets[s_v_index])[0];
```

```
            SV2[1] = (vSets[s_v_index])[1];
            SV2[2] = (vSets[s_v_index])[2];
            SV3[0] = (vSets[t_v_index])[0];
            SV3[1] = (vSets[t_v_index])[1];
            SV3[2] = (vSets[t_v_index])[2];

            //向量
            GLfloat vec1[3], vec2[3], vec3[3];
            vec1[0] = SV1[0] - SV2[0];
            vec1[1] = SV1[1] - SV2[1];
            vec1[2] = SV1[2] - SV2[2];
            vec2[0] = SV1[0] - SV3[0];
            vec2[1] = SV1[1] - SV3[1];
            vec2[2] = SV1[2] - SV3[2];
            vec3[0] = vec1[1] * vec2[2] - vec1[2] * vec2[1];
            vec3[1] = vec2[0] * vec1[2] - vec2[2] * vec1[0];
            vec3[2] = vec2[1] * vec1[0] - vec2[0] * vec1[1];

            GLfloat D = sqrt(pow(vec3[0], 2) + pow(vec3[1], 2) + pow(vec3[2], 2));
            VN[0] = vec3[0] / D;
            VN[1] = vec3[1] / D;
            VN[2] = vec3[2] / D;
            glNormal3f(VN[0], VN[1], VN[2]);

            glVertex3f(SV1[0], SV1[1], SV1[2]);
            glVertex3f(SV2[0], SV2[1], SV2[2]);
            glVertex3f(SV3[0], SV3[1], SV3[2]);
        }
        glEnd();
    }

    void ObjLoader::slice(int num_of_layers, int layer) //layer 为第几层截面
    {
        glBegin(GL_LINES);
        vector<GLfloat> export_obj;

        // 按照 z 轴平面切割
        float difference = (z_max - z_min) / num_of_layers;
        vector<GLfloat> S1{ 10000,10000,difference * layer + z_min };
        vector<GLfloat> S2{ -10000,10000,difference * layer + z_min };
        vector<GLfloat> S3{ 0,-10000,difference * layer + z_min };

        /*按照 y 轴平面切割
        float difference = (y_max - y_min) / 101;
```

```
vector<GLfloat> S1{ 10000,difference * layer + y_min,-10000 };
vector<GLfloat> S2{ -10000,difference * layer + y_min,-10000 };
vector<GLfloat> S3{ 0,difference * layer + y_min,10000 };
*/

/*按照 x 轴平面切割
float difference = (x_max - x_min) / 101;
vector<GLfloat> S1{ difference * layer + x_min,10000,-10000 };
vector<GLfloat> S2{ difference * layer + x_min,-10000,-10000 };
vector<GLfloat> S3{ difference * layer + x_min,0,10000 };
*/

Intersection cut_surface(S1, S2, S3);

for (int i = 0; i < fSets.size(); i++)
{
    GLint Index[3];
    for (int k = 0; k < 3; k++)
    {
        Index[k] = (fSets[i])[k];
    }
    vector<GLfloat> V1 = vSets[Index[0]], V2 = vSets[Index[1]], V3 = vSets[Index[2]];
    vector<GLfloat> node;
    vector<vector<GLfloat>> line;
    if (pan(V1, V2, cut_surface, node) == true)
        line.push_back(node);
    if (pan(V2, V3, cut_surface, node) == true)
        line.push_back(node);
    if (pan(V1, V3, cut_surface, node) == true)
        line.push_back(node);

    if (line.size() != 0)   //三角面片与切割面有交点
    {
        glVertex3f(line[0][0], line[0][1], line[0][2]);
        glVertex3f(line[1][0], line[1][1], line[1][2]);
        export_obj.push_back(line[0][0]);
        export_obj.push_back(line[0][1]);
        export_obj.push_back(line[0][2]);
        export_obj.push_back(line[1][0]);
        export_obj.push_back(line[1][1]);
        export_obj.push_back(line[1][2]);
    }
}
glEnd();
```

```
    //导出为 obj 文件
    string path = "F:\\ObjLoader\\output\\";  //切面文件的输出路径
    string fname = path + to_string(layer) + ".obj";
    ofstream out_obj;
    out_obj.open(fname, ios::out);
    for (int i = 0; i < export_obj.size() - 2; i += 3)
    {
        out_obj << "v" << " " << export_obj[i] << " " << export_obj[i + 1] << " " <<
export_obj[i + 2] << endl;
    }
    for (int j = 0; j < export_obj.size() - 1; j += 2)
    {
        out_obj << "e" << " " << j + 1 << " " << j + 2 << endl;
    }
    out_obj.close();
}
```

Intersection.cpp

```
#include "Intersection.h"
using namespace std;

Intersection::Intersection(vector<GLfloat> v_0, vector<GLfloat> v_1, vector<GLfloat> v_2)
{
    v0.resize(3);
    v1.resize(3);
    v2.resize(3);
    for (int i = 0; i < 3; i++) {
        v0[i] = v_0[i];
        v1[i] = v_1[i];
        v2[i] = v_2[i];
    }
}

GLfloat vector_dot(vector<GLfloat> v0, vector<GLfloat> v1)
{
    return v0[0] * v1[0] + v0[1] * v1[1] + v0[2] * v1[2];
}

void vector_minus(vector<GLfloat> a, vector<GLfloat> b, vector<GLfloat>& res)
{
    res.resize(3);
    res[0] = a[0] - b[0];
```

```
    res[1] = a[1] - b[1];
    res[2] = a[2] - b[2];
}

void vector_cross(vector<GLfloat> a, vector<GLfloat> b, vector<GLfloat>& res)
{
    res.resize(3);
    res[0] = a[1] * b[2] - a[2] * b[1];
    res[1] = a[2] * b[0] - a[0] * b[2];
    res[2] = a[0] * b[1] - a[1] * b[0];
}

bool pan(vector<GLfloat> start, vector<GLfloat> end, Intersection surface,
vector<GLfloat>& intersection)
{
    const float epsilon = 0.000001f;
    vector<GLfloat> e1, e2, p, s, q;
    float x, y, z, tmp;
    vector<GLfloat> direction;
    vector_minus(end, start, direction);
    vector_minus(surface.v1, surface.v0, e1);
    vector_minus(surface.v2, surface.v0, e2);
    vector_cross(direction, e2, p);
    tmp = vector_dot(p, e1);
    if (tmp > -epsilon && tmp < epsilon)
        return false;
    tmp = 1.0f / tmp;
    vector_minus(start, surface.v0, s);
    y = tmp * vector_dot(s, p);
    if (y < 0.0 || y > 1.0)
        return false;
    vector_cross(s, e1, q);
    z = tmp * vector_dot(direction, q);
    if (z < 0.0 || z > 1.0)
        return false;
    if (y + z > 1.0)
        return false;
    x = tmp * vector_dot(e2, q);
    if (x < 0.0 || x > 1.0)
        return false;
    intersection.resize(3);
    intersection[0] = start[0] + x * direction[0];
    intersection[1] = start[1] + x * direction[1];
    intersection[2] = start[2] + x * direction[2];
```

```
    return true;
}
```

main.cpp

```
#include "ObjLoader.h"
#include<iostream>
using namespace std;

string filePath = "data/monkey.obj";
ObjLoader objModel = ObjLoader(filePath);
static double c = 3.1415926 / 180.0f;
static double r = 1.0f;
static int degree = 90;
static int oldPosY = -1;
static int oldPosX = -1;

int layer_scan = 0;
int num_of_layers = 101;  //切割的总层数
string mode;

void setLightRes() {
    GLfloat lightPosition[] = { 0.0f, 0.0f, 1.0f, 0.0f };
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
}

void init() {
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("ObjLoader");
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    setLightRes();
    glEnable(GL_DEPTH_TEST);
}

void display_scan()
{
    glColor3f(1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
```

```
    glTranslatef(0.0f, 0.0f, -3.0f);
    setLightRes();
    glPushMatrix();
    gluLookAt(r * cos(c * degree), 0, r * sin(c * degree), 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
0.0f);
    objModel.slice(num_of_layers, layer_scan);
    if (layer_scan < num_of_layers-1) layer_scan++;
    glPopMatrix();
    glutSwapBuffers();
}

void display_model()
{
    glColor3f(1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -3.0f);
    setLightRes();
    glPushMatrix();
    gluLookAt(r * cos(c * degree), 0, r * sin(c * degree), 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
0.0f);
    objModel.Draw();
    glPopMatrix();
    glutSwapBuffers();
}

void display_select()
{
    glColor3f(1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -3.0f);
    setLightRes();
    glPushMatrix();
    gluLookAt(r * cos(c * degree), 0, r * sin(c * degree), 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
0.0f);
    //objModel.Draw();
    int layer;
    cout << "Enter the number of layers you want to view: ";
    cin >> layer;
    objModel.slice(num_of_layers, layer);
    glPopMatrix();
```

```
    glutSwapBuffers();
}

void reshape(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0f, (GLdouble)width / (GLdouble)height, 1.0f, 200.0f);
    glMatrixMode(GL_MODELVIEW);
}

void moseMove(int button, int state, int x, int y)
{
    if (state == GLUT_DOWN) {
        oldPosX = x; oldPosY = y;
    }
}

void changeViewPoint(int x, int y)
{
    int temp = x - oldPosX;
    degree += temp;
    oldPosX = x;
    oldPosY = y;
}

void myIdle()
{
    glutPostRedisplay();
}

int main(int argc, char* argv[])
{
    cout << "Choose the mode from 3Dmodel/scan/select: ";
    cin >> mode;
    if (mode == "scan")
    {
        for (int i = 0; i < num_of_layers-1; i++)
        {
            glutInit(&argc, argv);
            init();
            glutDisplayFunc(display_scan);
            glutReshapeFunc(reshape);
```

```
            glutMouseFunc(moseMove);
            glutMotionFunc(changeViewPoint);
            glutIdleFunc(myIdle);
            glutMainLoop();
            return 0;
            Sleep(0.001);
        }
    }
    else if (mode == "3Dmodel")
    {
        glutInit(&argc, argv);
        init();
        glutDisplayFunc(display_model);
        glutReshapeFunc(reshape);
        glutMouseFunc(moseMove);
        glutMotionFunc(changeViewPoint);
        glutIdleFunc(myIdle);
        glutMainLoop();
        return 0;
    }
    else if (mode == "select")
    {
        glutInit(&argc, argv);
        init();
        glutDisplayFunc(display_select);
        glutReshapeFunc(reshape);
        glutMouseFunc(moseMove);
        glutMotionFunc(changeViewPoint);
        glutIdleFunc(myIdle);
        glutMainLoop();
        return 0;
    }
}
```

5. 运行程序

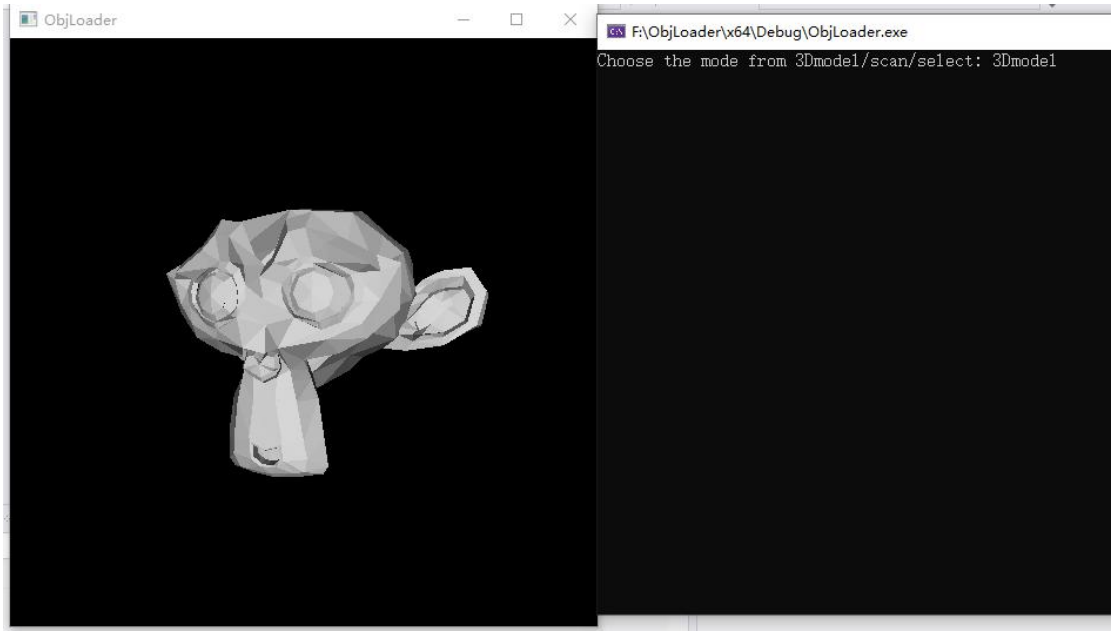
注意：在运行程序前建议新建一个 output 文件夹专门用于存放导出的切面文件。运行程序前需修改 ObjLoader.cpp 中的 slice 函数中的切面文件输出路径（path），将其修改为刚刚创建的 output 文件夹所在路径。

```
//导出为obj文件
string path = "F:\\ObjLoader\\output\\";  //切面文件的输出路径
string fname = path + to_string(layer) + ".obj";
ofstream out_obj;
```

点击运行程序，可从“3Dmodel”、“scan”和“select”三种模式中选择一种。

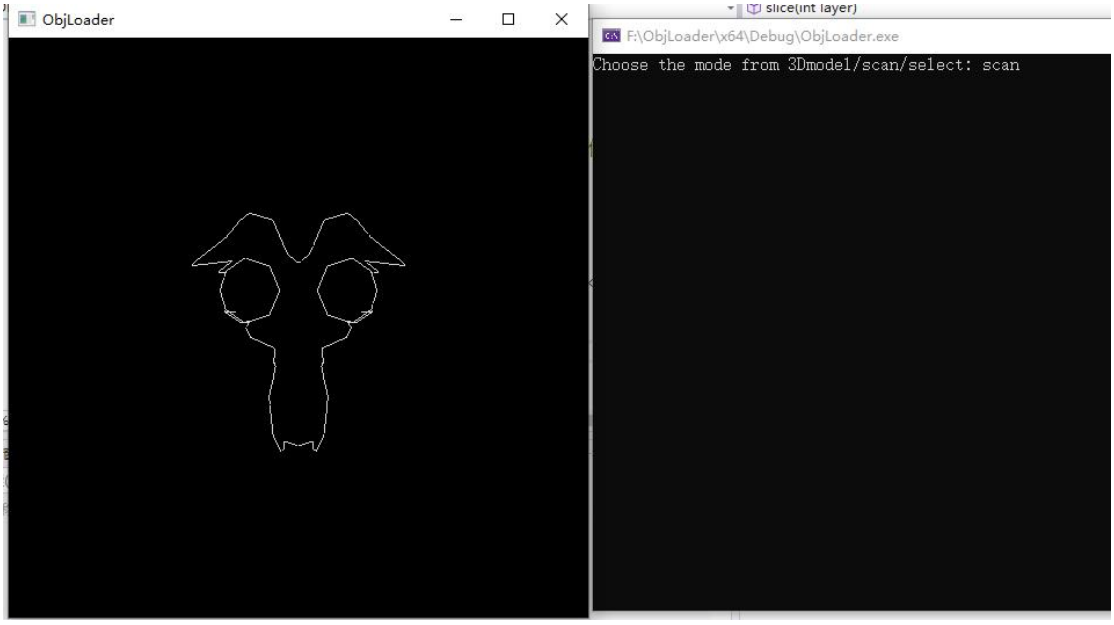
3Dmodel 模式

运行程序，输入“3Dmodel”,则显示 3D 模型，鼠标左右拖动可旋转查看模型。



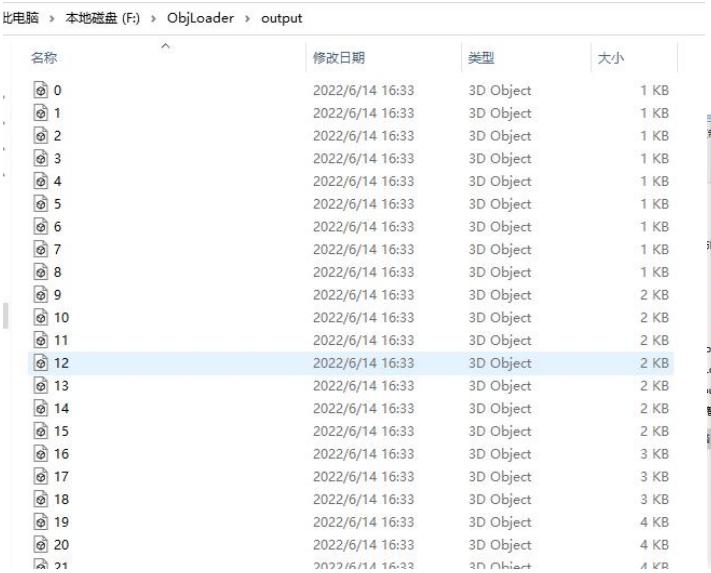
scan 模式

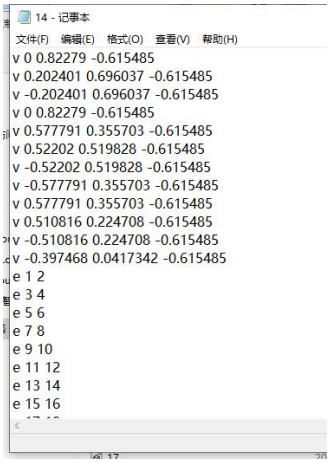
运行程序，输入“scan”,则会从第 0 层到第 100 依次显示切面，呈现动态的扫描效果。



程序在显示切面的同时，会将切面的 obj 格式文件输出到 output 文件夹中，可到文件

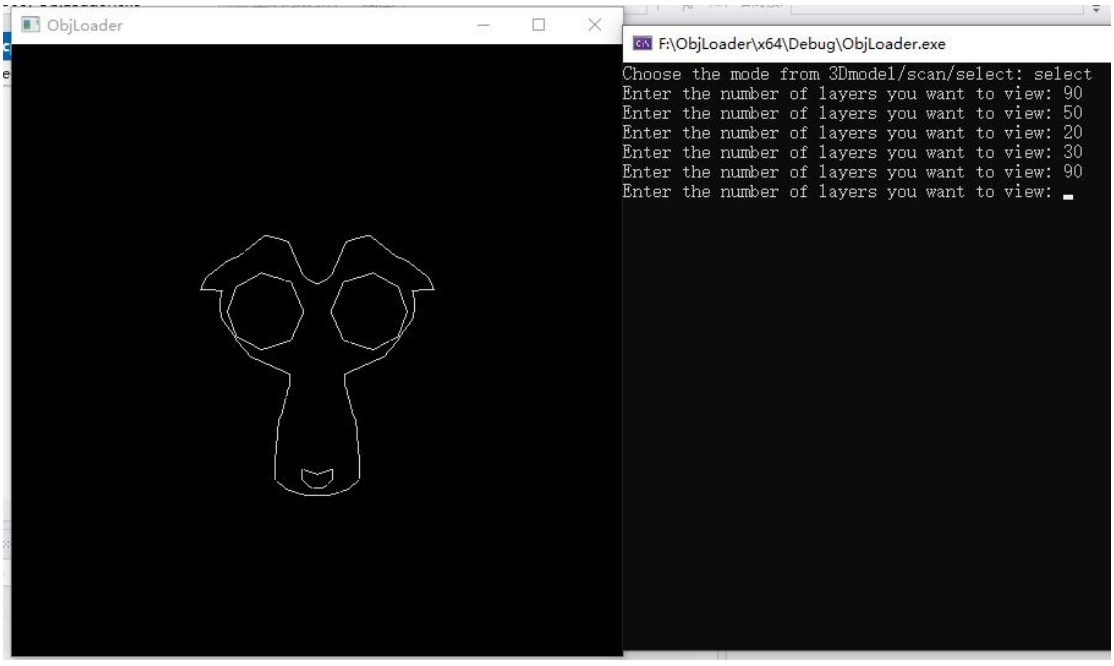
夹中查看各层的切面。（没有实现多边形填充，单纯的点和线段在 3D 场景中看不出来。但使用记事本打开可以看见保存的点和线的数据）





select 模式

运行程序，输入“select”,输入想要查看的层数，则会显示该层的切面（如输入 90，则会显示第 90 层的切面）。



同样的，程序在显示切面的同时，会将切面的 obj 格式文件输出到 output 文件夹中，可到文件夹中查看。

修改切割层数、修改切割平面（即切割方向）

1、切片的总层数可通过修改 main.cpp 中的 num_of_layers 改变，目前设置的是 100 层。

```
int layer_scan = 0;
int num_of_layers = 101;  //切割的总层数
string mode;
```

2、切割平面的方向可通过修改 ObjLoader.cpp 中的 slice 函数中的 S1, S2, S3 改变（切割平面本质上是一个大的三角面片，S1, S2, S3 为三角面片的三个顶点）。程序目前提供了 z 轴、y 轴、x 轴三个方向的等间距切片。

```
void ObjLoader::slice(int num_of_layers, int layer) //layer为第几层截面
{
    glBegin(GL_LINES);
    vector<GLfloat> export_obj;

    // 按照z轴平面切割
    float difference = (z_max - z_min) / num_of_layers;
    vector<GLfloat> S1{ 10000, 10000, difference * layer + z_min };
    vector<GLfloat> S2{ -10000, 10000, difference * layer + z_min };
    vector<GLfloat> S3{ 0, -10000, difference * layer + z_min };

    /*按照y轴平面切割
    float difference = (y_max - y_min) / 101;
    vector<GLfloat> S1{ 10000, difference * layer + y_min, -10000 };
    vector<GLfloat> S2{ -10000, difference * layer + y_min, -10000 };
    vector<GLfloat> S3{ 0, difference * layer + y_min, 10000 };
    */

    /*按照x轴平面切割
    float difference = (x_max - x_min) / 101;
    vector<GLfloat> S1{ difference * layer + x_min, 10000, -10000 };
```

6. 下一步工作建议（未来改进）

1、未能实现切片的多边形填充，所以导出的 obj 文件只有点和线段数据，在 3D 查看器中看不出来。未来考虑实现切片的填充，导出可以查看的切片文件。

2、目前运行程序时，主要通过键盘输入指令，与用户的交互不太友好。未来考虑用鼠标点击和简单的键盘按键来实现交互（考虑 MFC 和 OpenGL 结合，实现菜单栏等图形化界面）。

3、目前是通过设置切割平面的三点来修改切割平面，调整切割方向。程序目前提供 z 轴、y 轴、x 轴三个方向的切割，通过记录 z 轴、y 轴、x 轴的最大最小值，取其差值再除以切片层数来实现等间距切片（如 z_max - z_min / 100 可实现 z 轴方向等间距切 100 片）。

未来考虑实现任意方向的等间距切片。切割方向通过修改切割平面的三个顶点即可实现，关键在于如何实现等间距切片。目前思路是求出切割平面的法向量，求出 3D 模型在该法向量方向上的最大最小值，从而实现该方向的等间距切片。