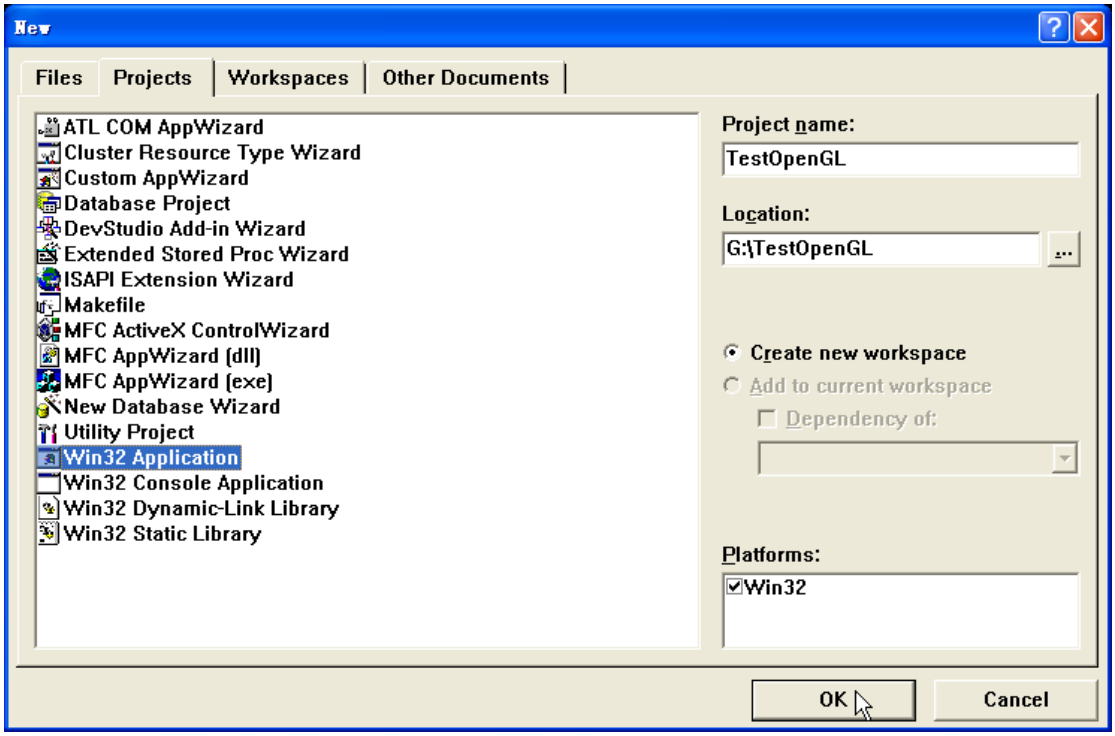
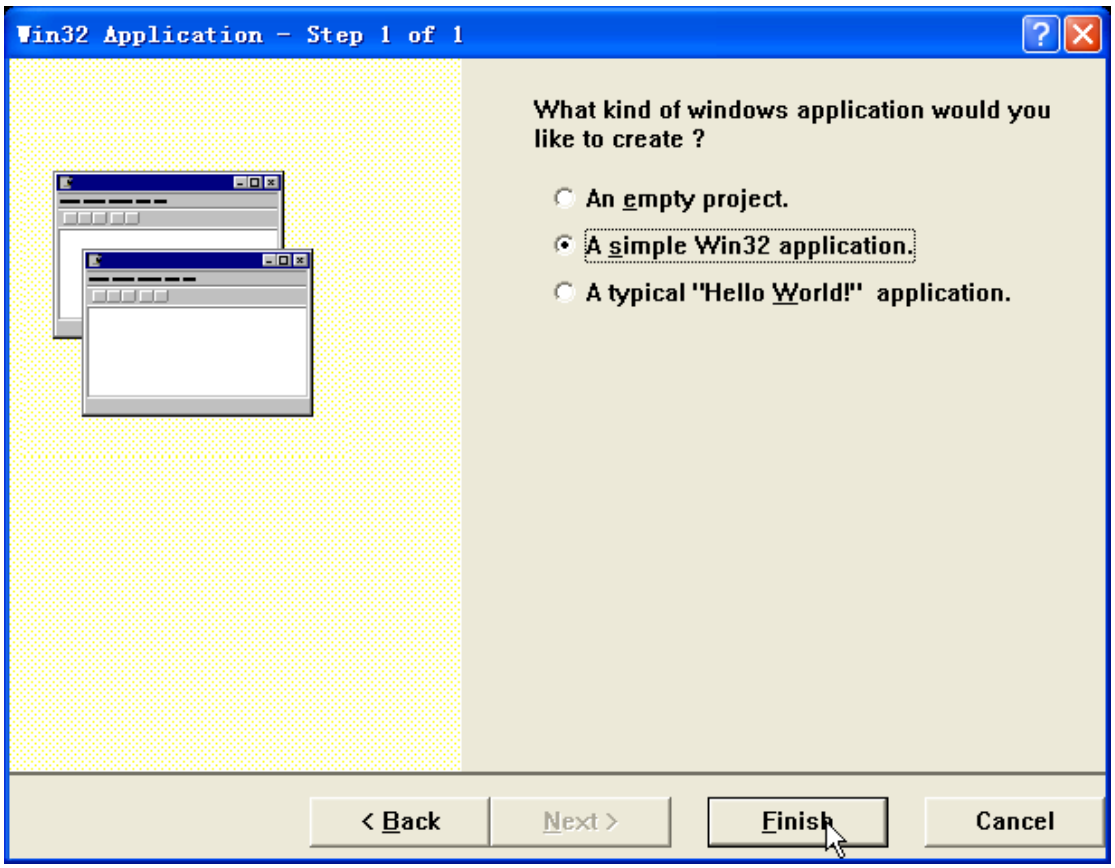
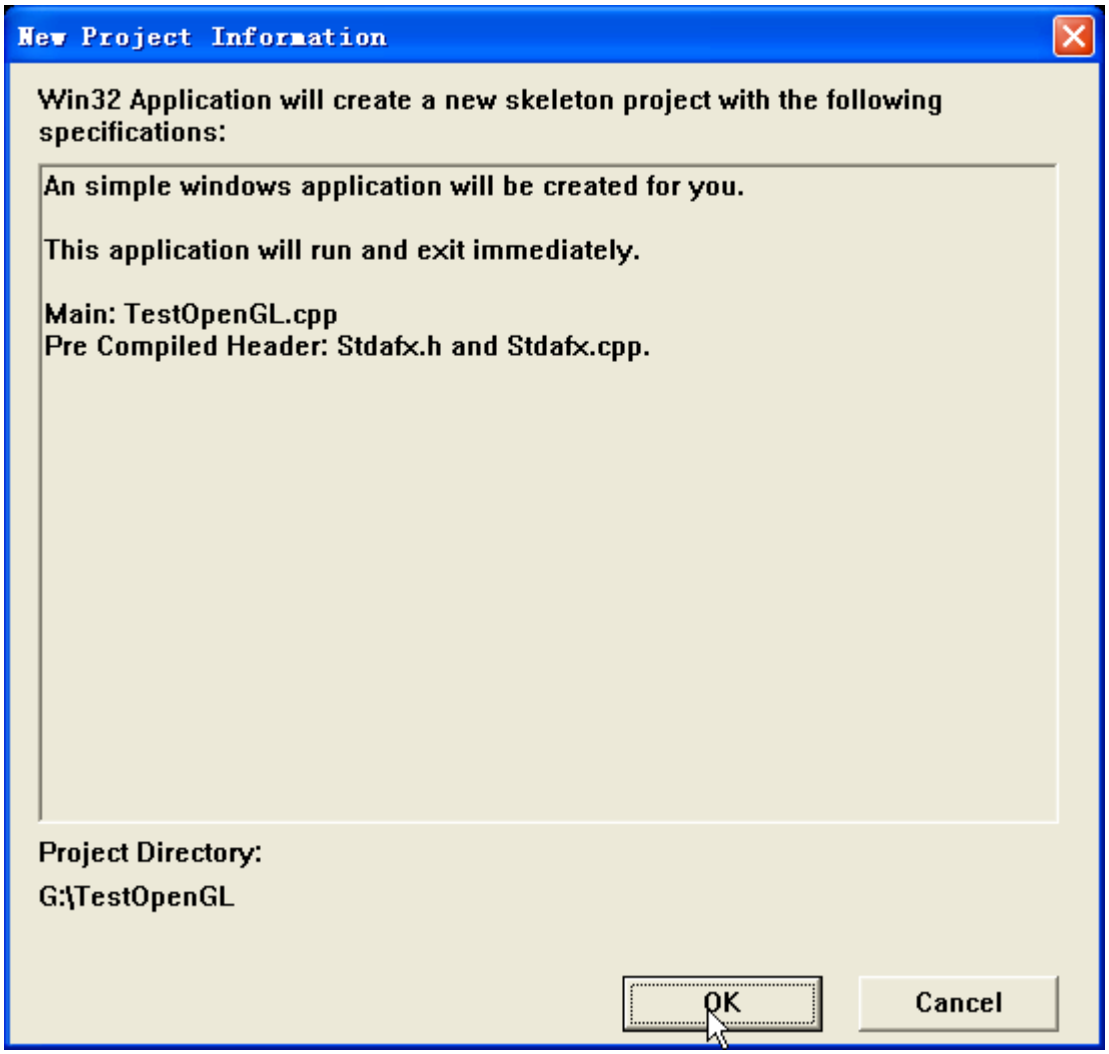
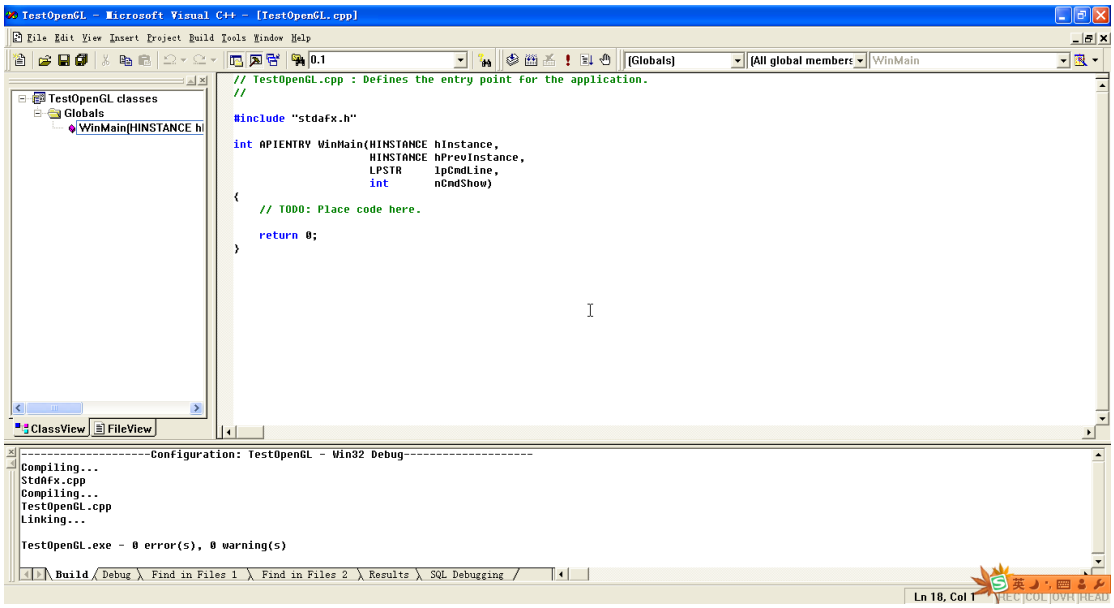
一．建立一个 Win32 应用程序

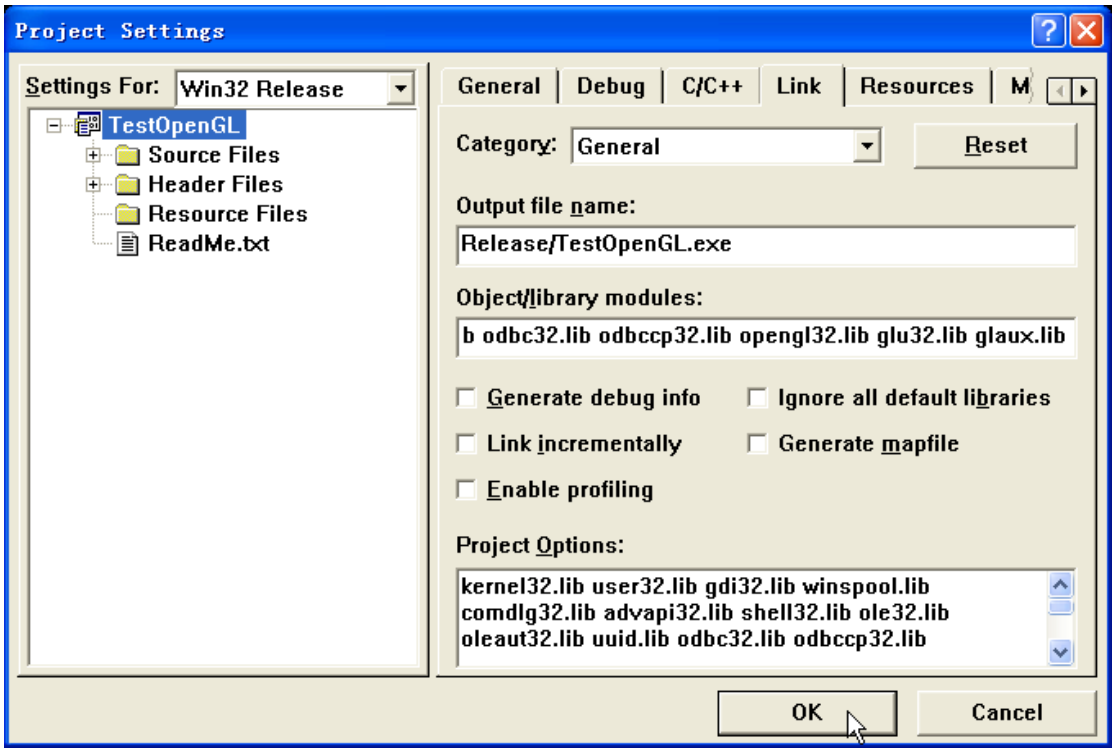








二．设置工程，使其可以进行 OpenGL 编程



三．包含头文件

```
#include "stdafx.h"
#include <gl\gl.h>          // 核心库
#include <gl\glu.h>         // 实用库
#include <gl\glaux.h>       // 辅助库
```

四．阅读 Nehe 的教程，编写程序框架，可以实现全屏显示或窗口显示

```
    return true;                                    // 一切OK
}

GLvoid KillGLWindow(GLvoid)                         // 正常销毁窗口
{
    if (fullscreen)                                 // 是否全屏模式?
    {
        ChangeDisplaySettings(NULL,0);              // 切换回桌面
        ShowCursor(true);                           // 显示鼠标指针
    }

    if (hRC)                                        // 有着色描述表吗?
    {
        if (!wglMakeCurrent(NULL,NULL))             // 我们能否释放DC和RC描述表?
        {
            MessageBox(NULL,"Release Of DC And RC Failed.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        }

        if (!wglDeleteContext(hRC))                 // 能否删除RC?
        {
            MessageBox(NULL,"Release Rendering Context Failed.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        }
        hRC=NULL;                                   // Set RC To NULL
    }

    if (hDC && !ReleaseDC(hWnd,hDC))                // 能否释放DC?
    {
        MessageBox(NULL,"Release Device Context Failed.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hDC=NULL;                                   // Set DC To NULL
    }

    if (hWnd && !DestroyWindow(hWnd))               // 能否销毁窗口?
    {
        MessageBox(NULL,"Could Not Release hWnd.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hWnd=NULL;                                  // Set hWnd To NULL
```

```
        MessageBox(NULL,"Could Not Release hWnd.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hWnd=NULL;                                      // Set hWnd To NULL
    }

    if (!UnregisterClass("OpenGL",hInstance))           // 能否注销类?
    {
        MessageBox(NULL,"Could Not Unregister Class.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hInstance=NULL;                                 // Set hInstance To NULL
    }
}

/*  This Code Creates Our OpenGL Window.  Parameters Are:                   *
 *  title           - Title To Appear At The Top Of The Window              *
 *  width           - Width Of The GL Window Or Fullscreen Mode             *
 *  height          - Height Of The GL Window Or Fullscreen Mode            *
 *  bits            - Number Of Bits To Use For Color (8/16/24/32)          *
 *  fullscreenflag  - Use Fullscreen Mode (true) Or Windowed Mode (false)   */

bool CreateGLWindow(char* title, int width, int height, int bits, bool fullscreenflag)
{
    GLuint      PixelFormat;            // 保存查找匹配的结果
    WNDCLASS    wc;                     // 窗口类结构
    DWORD       dwExStyle;              // 扩展窗口风格
    DWORD       dwStyle;                // 窗口风格
    RECT        WindowRect;             // 取得矩形的左上角和右下角的坐标值
    WindowRect.left=(long)0;            // Set Left Value To 0
    WindowRect.right=(long)width;       // Set Right Value To Requested Width
    WindowRect.top=(long)0;             // Set Top Value To 0
    WindowRect.bottom=(long)height;     // Set Bottom Value To Requested Height

    fullscreen=fullscreenflag;          // 设置全局全屏标记

    hInstance           = GetModuleHandle(NULL);                // 取得窗口的实例
    wc.style            = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;   // 移动时重画，并为窗口取得DC.
    wc.lpfnWndProc      = (WNDPROC) WndProc;                    // WndProc处理消息
    wc.hCursor          = LoadCursor(NULL, IDC_ARROW);          // 装入鼠标指针
    wc.hbrBackground    = NULL;                                 // GL不需要背景
    wc.lpszMenuName     = NULL;                                 // 不需要菜单
    wc.lpszClassName    = "OpenGL";                             // 设定类名字

    if (!RegisterClass(&wc))                                    // 尝试注册窗口类
    {
        MessageBox(NULL,"Failed To Register The Window Class.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return false;
    }

    if (fullscreen)                                             // 尝试全屏模式?
    {
        DEVMODE dmScreenSettings;                               // 设备模式
        memset(&dmScreenSettings,0,sizeof(dmScreenSettings));   // 确保内存分配
        dmScreenSettings.dmSize=sizeof(dmScreenSettings);       // Size Of The Devmode Structure
        dmScreenSettings.dmPelsWidth    = width;                // Selected Screen Width
        dmScreenSettings.dmPelsHeight   = height;               // Selected Screen Height
        dmScreenSettings.dmBitsPerPel   = bits;                 // Selected Bits Per Pixel
        dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;

        // 尝试设置显示模式并返回结果。注: CDS_FULLSCREEN 移去了状态条。
        if (ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=DISP_CHANGE_SUCCESSFUL)
        {
            // 若模式失败，提供两个选项，退出或在窗口内运行
            if (MessageBox(NULL,"The Requested Fullscreen Mode Is Not Supported By\nYour Video Card. Use Windowed Mode Instead?","NeH
            {
                fullscreen=false;       // 选择窗口模式
            }
            else
            {
                // 如果用户选择退出，弹出消息窗口告知用户程序将结束。
                MessageBox(NULL,"Program Will Now Close.","ERROR",MB_OK|MB_ICONSTOP);
                return false;                               // Return False
            }

                return false;                               // Return False
            }
        }
    }

    if (fullscreen)                                         // 仍处于全屏模式?
    {
        dwExStyle=WS_EX_APPWINDOW;                          // 扩展窗体风格
        dwStyle=WS_POPUP;                                   // 窗体风格
        ShowCursor(false);                                  // 隐藏鼠标指针
    }
    else
    {
        dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;       // 扩展窗体风格
        dwStyle=WS_OVERLAPPEDWINDOW;                        // 窗体风格
    }

    AdjustWindowRectEx(&WindowRect, dwStyle, false, dwExStyle);     // Adjust Window To True Requested Size

    // Create The Window
    if (!(hWnd=CreateWindowEx(  dwExStyle,                          // Extended Style For The Window
                                "OpenGL",                           // Class Name
                                title,                              // Window Title
                                dwStyle |                           // Defined Window Style
                                WS_CLIPSIBLINGS |                   // Required Window Style
                                WS_CLIPCHILDREN,                    // Required Window Style
                                0, 0,                               // Window Position
                                WindowRect.right-WindowRect.left,   // Calculate Window Width
                                WindowRect.bottom-WindowRect.top,   // Calculate Window Height
                                NULL,                               // No Parent Window
                                NULL,                               // No Menu
                                hInstance,                          // Instance
                                NULL)))                             // Dont Pass Anything To WM_CREATE
    {
        KillGLWindow();                             // Reset The Display
```

```
        return false;                                   // Return false
    }

    static  PIXELFORMATDESCRIPTOR pfd=                  // pfd Tells Windows How We Want Things To Be
    {
        sizeof(PIXELFORMATDESCRIPTOR),                  // Size Of This Pixel Format Descriptor
        1,                                              // Version Number
        PFD_DRAW_TO_WINDOW |                            // Format Must Support Window
        PFD_SUPPORT_OPENGL |                            // Format Must Support OpenGL
        PFD_DOUBLEBUFFER,                               // Must Support Double Buffering
        PFD_TYPE_RGBA,                                  // Request An RGBA Format
        bits,                                           // Select Our Color Depth
        0, 0, 0, 0, 0, 0,                               // Color Bits Ignored
        0,                                              // No Alpha Buffer
        0,                                              // Shift Bit Ignored
        0,                                              // No Accumulation Buffer
        0, 0, 0, 0,                                     // Accumulation Bits Ignored
        16,                                             // 16Bit Z-Buffer (Depth Buffer)
        0,                                              // No Stencil Buffer
        0,                                              // No Auxiliary Buffer
        PFD_MAIN_PLANE,                                 // Main Drawing Layer
        0,                                              // Reserved
        0, 0, 0                                         // Layer Masks Ignored
    };

    if (!(hDC=GetDC(hWnd)))                             // 取得设备描述表了么?
    {
        KillGLWindow();                                 // Reset The Display
        MessageBox(NULL,"Can't Create A GL Device Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return false;                                   // Return false
    }

    if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))     // Did Windows Find A Matching Pixel Format?
    {
        KillGLWindow();                                 // Reset The Display
```

```
    if (!(hRC=wglCreateContext(hDC)))                   // Are We Able To Get A Rendering Context?
    {
        KillGLWindow();                                 // Reset The Display
        MessageBox(NULL,"Can't Create A GL Rendering Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return false;                                   // Return false
    }

    if(!wglMakeCurrent(hDC,hRC))                        // Try To Activate The Rendering Context
    {
        KillGLWindow();                                 // Reset The Display
        MessageBox(NULL,"Can't Activate The GL Rendering Context.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return false;                                   // Return false
    }

    ShowWindow(hWnd,SW_SHOW);                           // 显示窗口
    SetForegroundWindow(hWnd);                          // Slightly Higher Priority
    SetFocus(hWnd);                                     // 设置键盘的焦点至窗口
    ReSizeGLScene(width, height);                       // 设置透视GL屏幕

    if (!InitGL())                                      // Initialize Our Newly Created GL Window
    {
        KillGLWindow();                                 // Reset The Display
        MessageBox(NULL,"Initialization Failed.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return false;                                   // Return false
    }

    return true;                                        // Success
}

LRESULT CALLBACK WndProc(   HWND    hWnd,           // Handle For This Window
                            UINT    uMsg,           // Message For This Window
                            WPARAM  wParam,         // Additional Message Information
                            LPARAM  lParam)         // Additional Message Information
```

```
        return false;                                   // Return false
    }

    return true;                                        // Success
}

LRESULT CALLBACK WndProc(   HWND    hWnd,           // Handle For This Window
                            UINT    uMsg,           // Message For This Window
                            WPARAM  wParam,         // Additional Message Information
                            LPARAM  lParam)         // Additional Message Information
{
    switch (uMsg)                                       // Check For Windows Messages
    {
        case WM_ACTIVATE:                               // Watch For Window Activate Message
        {
            if (!HIWORD(wParam))                        // Check Minimization State
            {
                active=true;                            // Program Is Active
            }
            else
            {
                active=false;                           // Program Is No Longer Active
            }

            return 0;                                   // Return To The Message Loop
        }

        case WM_SYSCOMMAND:                             // Intercept System Commands
        {
            switch (wParam)                             // Check System Calls
            {
                case SC_SCREENSAVE:                     // Screensaver Trying To Start?
                case SC_MONITORPOWER:                   // Monitor Trying To Enter Powersave?
                return 0;                               // Prevent From Happening
```

```
            {
                case SC_SCREENSAVE:                 // Screensaver Trying To Start?
                case SC_MONITORPOWER:               // Monitor Trying To Enter Powersave?
                return 0;                           // Prevent From Happening
            }
            break;                                  // Exit
        }

        case WM_CLOSE:                              // Did We Receive A Close Message?
        {
            PostQuitMessage(0);                     // Send A Quit Message
            return 0;                               // Jump Back
        }

        case WM_KEYDOWN:                            // Is A Key Being Held Down?
        {
            keys[wParam] = true;                    // If So, Mark It As true
            return 0;                               // Jump Back
        }

        case WM_KEYUP:                              // Has A Key Been Released?
        {
            keys[wParam] = false;                   // If So, Mark It As false
            return 0;                               // Jump Back
        }

        case WM_SIZE:                               // Resize The OpenGL Window
        {
            ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));  // LoWord=Width, HiWord=Height
            return 0;                               // Jump Back
        }
    }

    // Pass All Unhandled Messages To DefWindowProc
    return DefWindowProc(hWnd,uMsg,wParam,lParam);
}

int WINAPI WinMain( HINSTANCE   hInstance,          // 实例
                    HINSTANCE   hPrevInstance,      // 前一个实例
                    LPSTR       lpCmdLine,          // 命令行参数
                    int         nCmdShow)           // 窗口显示状态
{
    MSG     msg;                                    // Windows Message Structure
    bool    done=false;                             // Bool Variable To Exit Loop

    nFullWidth = GetSystemMetrics(SM_CXSCREEN);
    nFullHeight = GetSystemMetrics(SM_CYSCREEN);

    // Ask The User Which Screen Mode They Prefer
    if (MessageBox(NULL,"全屏运行吗?", "选择运行模式",MB_YESNO|MB_ICONQUESTION)==IDNO)
    {
        fullscreen=false;                           // 窗口模式
    }

    // Create Our OpenGL Window
    if (!CreateGLWindow("机器人",nFullWidth,nFullHeight,32,fullscreen))
    {
        return 0;                                   // Quit If Window Was Not Created
    }
    Reset();

    while(!done)
    {
        if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))   // Is There A Message Waiting?
        {
            return 0;                               // Quit If Window Was Not Created
    }
    Reset();

    while(!done)
    {
        if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))   // Is There A Message Waiting?
        {
            if (msg.message==WM_QUIT)               // Have We Received A Quit Message?
            {
                glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                glClearDepth(1.0);
                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
                done=true;                          // If So done=true
            }
            else                                    // If Not, Deal With Window Messages
            {
                TranslateMessage(&msg);             // Translate The Message
                DispatchMessage(&msg);              // Dispatch The Message
            }
        }
        else                                        // If There Are No Messages
        {
            // Draw The Scene.  Watch For ESC Key And Quit Messages From DrawGLScene()
            if (active)                             // Program Active?
            {
                if (keys[VK_ESCAPE])                // Was ESC Pressed?
                {
                    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
                    glClearDepth(1.0);
                    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
                    done=true;                      // ESC Signalled A Quit
                }
                else                                // Not Time To Quit, Update Screen
```

五．在绘图函数中添加代码，通过计算坐标，绘制出一个漂亮的机器人形体

```
    return true;                                    // 初始化 OK
}

//绘图函数
int DrawGLScene(GLvoid)
{
    //设置清除屏幕的颜色，并清除屏幕和深度缓冲
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    G_vLitPosition[0] = 5.0f + G_fLightx;

    ////////////////////////////////////////////////
    //设置光照
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glLightfv(GL_LIGHT0, GL_AMBIENT, G_vLitAmbient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, G_vLitDiffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, G_vLitSpecular);
    glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 3.0f);
    glLightfv(GL_LIGHT0, GL_POSITION, G_vLitPosition);
    glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, 1.0);

    glColorMaterial(GL_FRONT, GL_DIFFUSE);
    glEnable(GL_COLOR_MATERIAL);
    //打开光照
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    ////////////////////////////////////////////////
    //变换并绘制物体
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    //坐标中心向Z轴平移-G_fDistance (使坐标中心位于摄像机前方)
    glTranslatef(G_fDistancce_horizon, G_fDistancce_vertical, -G_fDistance);
//  glTranslatef(G_fDistancce_horizon, 0.0, 0.0);
//  glTranslatef(0.0, G_fDistancce_vertical, 0.0);
    glRotatef(G_fAngle_horizon, 0.0f, 1.0f, 0.0f);
    glRotatef(G_fAngle_vertical, 1.0f, 0.0f, 0.0f);

    ////////////////////////////////////////////////////////////////////////////////////////////////
    //脑袋
glPushMatrix();
    glTranslatef(0, 1, 0);
    glRotatef(G_fHead_Ang, 0, 1, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.5);
    else
        auxSolidSphere(0.5);

    ////////////////////////////////////////////////////////////////////////////////////////////////
    //鼻子
glPushMatrix();
    glTranslatef(0, -0.1f, 0.5f);
    glColor3f(0.973f, 0.973f, 1.0f);
    if(wire)
        auxWireBox(0.05, 0.15, 0.05);
    else
        auxSolidBox(0.05, 0.15, 0.05);

    ////////////////////////////////////////////////////////////////////////////////////////////////
    //嘴巴
```

```
    //嘴巴
    glTranslatef(0, -0.2f, -0.1f);
    glColor3f(1.0f, 0.753f, 0.796f);
    if(wire)
        auxWireBox(0.1f, 0.02f, 0.05f);
    else
        auxSolidBox(0.1f, 0.02f, 0.05f);

    ////////////////////////////////////////////////////////////////////////////////////////////////
    //眼睛
glPushMatrix();
    glColor3f(0.118f, 0.565f, 1.0f);
    glTranslatef(-0.15f, 0.4f, 0.07f);
    if(wire)
        auxWireSphere(0.05);
    else
        auxSolidSphere(0.05);
    glTranslatef(0.3f, 0.0f, 0.0f);
    if(wire)
        auxWireSphere(0.05);
    else
        auxSolidSphere(0.05);
glPopMatrix();
glPopMatrix();
glPopMatrix();

glPushMatrix();
    ////////////////////////////////////////////////////////////////////////////////////////////////
    //身体
    //glTranslatef(0.0f, 0.0f, -0.2f);
    glRotatef(90, 0, 1, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireIcosahedron(0.75);
    else
```

```
glPopMatrix();
glPushMatrix();

    ////////////////////////////////////////////////////////////////////////////////////////////////
    //右手
    glTranslatef(0.65f, 0.0f, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.1);
    else
        auxSolidSphere(0.1);
    glRotatef(G_fRArmF_Ang, 1, 0, 0);
    glRotatef(G_fRArm_Ang + 180, 0, 0, -1);
    glTranslatef(0, -0.5, 0);
    glColor3f(MYCOLOR);

    if(wire)
        auxWireCylinder(0.1, 0.5);
    else
        auxSolidCylinder(0.1, 0.5);
    glTranslatef(0.0, 1.0, 0.0);
    glColor3f(MYCOLOR);

    if(wire)
        auxWireSphere(0.1);
    else
        auxSolidSphere(0.1);

    glRotatef(-G_fRArm_Ang + G_fRDArm_Ang, 0, 0, -1); //侧抬降右下臂
    glTranslatef(0, -0.5, 0);
    glColor3f(MYCOLOR);

    if(wire)
```

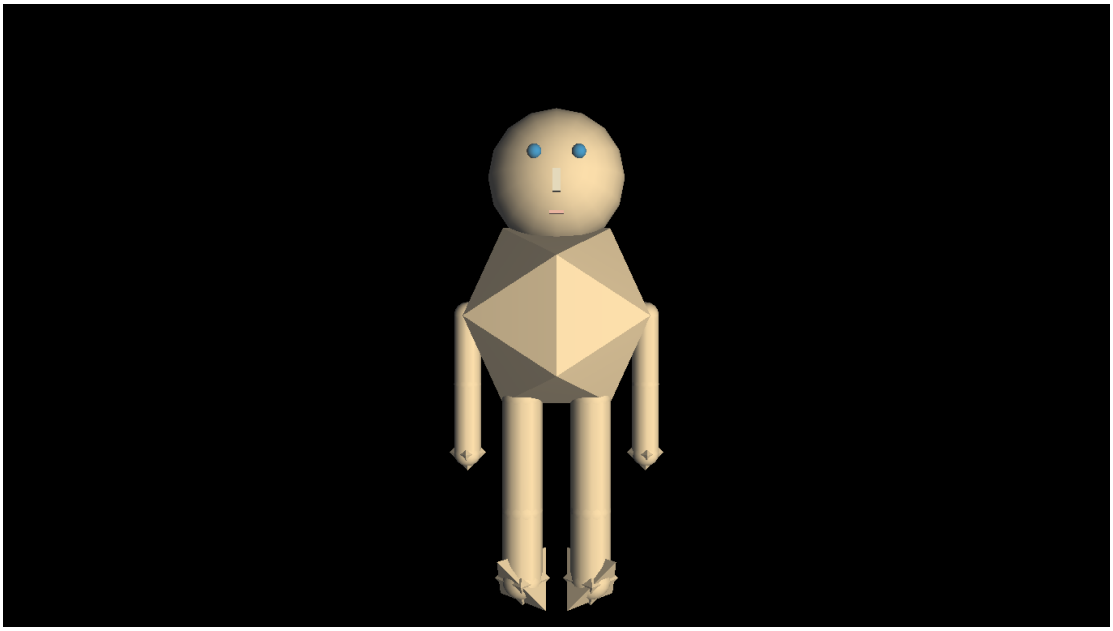
```
        auxSolidCylinder(0.1, 0.5);
    glTranslatef(0, 1, 0);
    glColor3f(MYCOLOR);

    if(wire)
    {
        auxWireSphere(0.1);
        auxWireOctahedron(0.135);
    }
    else
    {
        auxSolidSphere(0.1);
        auxSolidOctahedron(0.135);
    }
glPopMatrix();

glPushMatrix();

    ////////////////////////////////////////////////////////////////////////////////////////////////
    //左手
    glColor3f(MYCOLOR);
    glTranslatef(-0.65f, 0.0f, 0);
    if(wire)
        auxWireSphere(0.1);
    else
        auxSolidSphere(0.1);
    glRotatef(G_fLArmF_Ang, 1, 0, 0); //前后抬手臂
    glRotatef(G_fLArm_Ang - 180, 0, 0, -1); //侧抬降左上臂
    glTranslatef(0, -0.5, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireCylinder(0.1, 0.5);
    else
        auxSolidCylinder(0.1, 0.5);
```

```
    if(wire)
        auxWireSphere(0.1);
    else
        auxSolidSphere(0.1);
    glRotatef(-G_fLArm_Ang + G_fLDArm_Ang, 0, 0, -1); //側抬降左下臂
    glTranslatef(0, -0.5, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireCylinder(0.1, 0.5);
    else
        auxSolidCylinder(0.1, 0.5);
    glTranslatef(0, 1, 0);
    glColor3f(MYCOLOR);
    if(wire)
    {
        auxWireSphere(0.1);
        auxWireOctahedron(0.135);
    }
    else
    {
        auxSolidSphere(0.1);
        auxSolidOctahedron(0.135);
    }
glPopMatrix();

glPushMatrix();
    ////////////////////////////////////////////////////////////////////////////////////////////
    //右腿
    glTranslatef(0.25f, -0.6f, 0.0f);
    glRotatef(G_fRLeg_Horizon, 0, 0, -1);
    glRotatef(180 + G_fRLeg_Ang, 1.0f, 0.0f, 0.0f); //抬右大腿
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.15);
    else
```

```
    //右腿
    glTranslatef(0.25f, -0.6f, 0.0f);
    glRotatef(G_fRLeg_Horizon, 0, 0, -1);
    glRotatef(180 + G_fRLeg_Ang, 1.0f, 0.0f, 0.0f); //抬右大腿
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.15);
    else
        auxSolidSphere(0.15);
    glTranslatef(0, -0.18f, 0);
    glColor3f(MYCOLOR);

    if(wire)
        auxWireCylinder(0.15, 0.8);
    else
        auxSolidCylinder(0.15, 0.8);
    glTranslatef(0, 1, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.15);
    else
        auxSolidSphere(0.15);
    glRotatef(-G_fRLeg_Ang + G_fRCalf_Ang, 1.0, 0.0, 0.0); //抬右小腿
    glTranslatef(0, -0.5, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireCylinder(0.15, 0.5);
    else
        auxSolidCylinder(0.15, 0.5);
    glTranslatef(0, 1, 0);
    glColor3f(MYCOLOR);
    if(wire)
    {
        auxWireSphere(0.15);
```

```
    glColor3f(MYCOLOR);
    if(wire)
    {
        auxWireSphere(0.15);
        auxWireOctahedron(0.202);
    }
    else
    {
        auxSolidSphere(0.15);
        auxSolidOctahedron(0.202);
    }
    glRotatef(180, 0, 1, 0);

    if(wire)
    {
        auxWireCone(0.15, 0.15);
        auxWireTetrahedron(0.3);
    }
    else
    {
        auxSolidCone(0.15, 0.15);
        auxSolidTetrahedron(0.3);
    }
glPopMatrix();

glPushMatrix();
    ////////////////////////////////////////////////////////////////////////////////////////////
    //左腿
    glTranslatef(-0.25f, -0.6f, 0.0f);
    glRotatef(G_fLLeg_Horizon, 0, 0, -1);
    glRotatef(180 + G_fLLeg_Ang, 1.0f, 0.0f, 0.0f); //抬左大腿
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.15);
```

```
    //左腿
    glTranslatef(-0.25f, -0.6f, 0.0f);
    glRotatef(G_fLLeg_Horizon, 0, 0, -1);
    glRotatef(180 + G_fLLeg_Ang, 1.0f, 0.0f, 0.0f); //抬左大腿
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.15);
    else
        auxSolidSphere(0.15);
    glTranslatef(0, -0.18f, 0);
    glColor3f(MYCOLOR);

    if(wire)
        auxWireCylinder(0.15, 0.8);
    else
        auxSolidCylinder(0.15, 0.8);
    glTranslatef(0, 1, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireSphere(0.15);
    else
        auxSolidSphere(0.15);
    glRotatef(-G_fLLeg_Ang + G_fLCalf_Ang, 1.0, 0.0, 0.0); //抬右小腿
    glTranslatef(0, -0.5, 0);
    glColor3f(MYCOLOR);
    if(wire)
        auxWireCylinder(0.15, 0.5);
    else
        auxSolidCylinder(0.15, 0.5);
    glTranslatef(0, 1, 0);
    glColor3f(MYCOLOR);
    if(wire)
    {
        auxWireSphere(0.15);
        auxWireOctahedron(0.202);
    }
    else
    {
        auxSolidSphere(0.15);
        auxSolidOctahedron(0.202);
    }
    glRotatef(90, 0, 1, 0);

    if(wire)
    {
        auxWireCone(0.15, 0.15);
        auxWireTetrahedron(0.3);
    }
    else
    {
        auxSolidCone(0.15, 0.15);
        auxSolidTetrahedron(0.3);
    }
glPopMatrix();

    auxSwapBuffers();

    return true;                                    // 一切OK
}

GLvoid KillGLWindow(GLvoid)                         // 正常销毁窗口
```



六．添加函数，修改绘制机器人过程中的各个角度参数，并响应键盘消息，实现对机器人形状的控制

```
int nFullWidth;
int nFullHeight;

//////////////////////////////////////////////////////////////////////////////////////////
//远近距离
float G_fDistance;

//水平移动距离
float G_fDistancce_horizon;

//垂直移动距离
float G_fDistancce_vertical;

//水平旋转角度
float G_fAngle_horizon;

//垂直旋转角度
float G_fAngle_vertical;

//右上臂旋转角度
float G_fRArm_Ang;

//左上臂旋转角度
float G_fLArm_Ang;

//右大腿旋转角度
float G_fRLeg_Ang;

//左大腿旋转角度
float G_fLLeg_Ang;

//右小腿旋转角度
float G_fRCalf_Ang;

//左小腿旋转角度

//右上臂旋转角度
float G_fRArm_Ang;

//左上臂旋转角度
float G_fLArm_Ang;

//右大腿旋转角度
float G_fRLeg_Ang;

//左大腿旋转角度
float G_fLLeg_Ang;

//右小腿旋转角度
float G_fRCalf_Ang;

//左小腿旋转角度
float G_fLCalf_Ang;

//左下臂旋转角度
float G_fLDArm_Ang;

//右下臂旋转角度
float G_fRDArm_Ang;

//左上臂旋转角度
float G_fLArmF_Ang;

//右上臂旋转角度
float G_fRArmF_Ang;

//脑袋旋转角度
float G_fHead_Ang;

//右臂水平偏转角

//左上臂旋转角度
float G_fLArmF_Ang;

//右上臂旋转角度
float G_fRArmF_Ang;

//脑袋旋转角度
float G_fHead_Ang;

//右臂水平偏转角
float G_fRLeg_Horizon;

//左臂水平偏转角
float G_fLLeg_Horizon;

//光照角度
float G_fLightx;

//光照参数
float G_vLitAmbient[4] = { 0.8f, 0.8f, 0.8f, 1.0f };

float G_vLitDiffuse[4] = { 0.8f, 0.75f, 0.6f, 1.0f };

float G_vLitSpecular[4] = { 0.5f, 0.5f, 0.5f, 1.0f };

float G_vLitPosition[4] = { 5.0f, 0.0f, 5.0f, 1.0f };

float G_vMaterialSpecu[4] = { 0.0f, 0.0f, 0.0f, 1.0f };

//////////////////////////////////////////////////////////////////////////////////////////
//函数声明
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

//初始化
```

```
    // Shutdown
    KillGLWindow();                                 // Kill The Window
    return (msg.wParam);                            // Exit The Program
}

//移近
void inline MoveNear(void)
{
    G_fDistance -= 0.1f;
}

//移远
void inline MoveFar(void)
{
    G_fDistance += 0.1f;
}

//左移
void inline MoveLeft(void)
{
    G_fDistancce_horizon -= 0.02f;
}

//右移
void inline MoveRight(void)
{
    G_fDistancce_horizon += 0.02f;
}

//上移
void inline MoveUp(void)
{
    G_fDistancce_vertical += 0.05f;
}
```


```
//上移
void inline MoveUp(void)
{
    G_fDistancce_vertical += 0.05f;
}
```


```
//下移
void inline MoveDown(void)
{
    G_fDistancce_vertical -= 0.05f;
}

//左转
void inline LeftRotate(void)
{
    G_fAngle_horizon -= 5;
}

//右转
void inline RightRotate(void)
{
    G_fAngle_horizon += 5;
}

//向上转动
void inline UpRotate(void)
{
    G_fAngle_vertical -= 5;
}

//向下转动
void inline DownRotate(void)
{
    G_fAngle_vertical += 5;
}
```


```
    G_fAngle_vertical -= 5;
}

//向下转动
void inline DownRotate(void)
{
    G_fAngle_vertical += 5;
}
```


```
//左上臂向下转动
void inline RotateLeftArmDown(void)
{
    if (G_fLArm_Ang > 0)
    {
        G_fLArm_Ang -= 3;
    }
    if (G_fLArm_Ang < G_fLDArm_Ang)
    {
        G_fLDArm_Ang = G_fLArm_Ang;
    }
}

//左上臂向上转动
void inline RotateLeftArmUp(void)
{
    if (G_fLArm_Ang < 135)
    {
        G_fLArm_Ang += 3;
    }
    if (G_fLArm_Ang - 135 > G_fLDArm_Ang)
    {
        G_fLDArm_Ang = G_fLArm_Ang - 135;
    }
}
```

```
}

//右上臂向下转动
void inline RotateRightArmDown(void)
{
    if (G_fRArm_Ang < 0)
    {
        G_fRArm_Ang += 3;
    }
    if (G_fRArm_Ang > G_fRDArm_Ang)
    {
        G_fRDArm_Ang = G_fRArm_Ang;
    }
}

//右上臂向上转动
void inline RotateRightArmUp(void)
{
    if (G_fRArm_Ang > -135)
    {
        G_fRArm_Ang -= 3;
    }
    if (G_fRArm_Ang + 135 < G_fRDArm_Ang)
    {
        G_fRDArm_Ang = G_fRArm_Ang + 135;
    }
}

//左大腿向下旋转
void inline RotateLeftLegDown(void)
{
    if (G_fLLeg_Ang < 90)
        G_fLLeg_Ang += 3;
    if (G_fLLeg_Ang > G_fLCalf_Ang)
```

```
//左大腿向下旋转
void inline RotateLeftLegDown(void)
{
    if (G_fLLeg_Ang < 90)
        G_fLLeg_Ang += 3;
    if (G_fLLeg_Ang > G_fLCalf_Ang)
        G_fLCalf_Ang = G_fLLeg_Ang;
}

//左大腿向上旋转
void inline RotateLeftLegUp(void)
{
    if (G_fLLeg_Ang > -135)
        G_fLLeg_Ang -= 3;
    if (G_fLCalf_Ang > G_fLLeg_Ang + 135)
        G_fLCalf_Ang = G_fLLeg_Ang + 135;
}

//右大腿向下旋转
void inline RotateRightLegDown(void)
{
    if (G_fRLeg_Ang < 90)
        G_fRLeg_Ang += 3;
    if (G_fRLeg_Ang > G_fRCalf_Ang)
        G_fRCalf_Ang = G_fRLeg_Ang;
}

//右大腿向上旋转
void inline RotateRightLegUp(void)
{
    if (G_fRLeg_Ang > -135)
        G_fRLeg_Ang -= 3;
    if (G_fRCalf_Ang > G_fRLeg_Ang + 135)
```

```
//右大腿向上旋转
void inline RotateRightLegUp(void)
{
    if (G_fRLeg_Ang > -135)
        G_fRLeg_Ang -= 3;
    if (G_fRCalf_Ang > G_fRLeg_Ang + 135)
        G_fRCalf_Ang = G_fRLeg_Ang + 135;
}

//右小腿向下旋转
void inline RotateRightCalfDown(void)
{
    if (G_fRCalf_Ang < G_fRLeg_Ang + 135)
        G_fRCalf_Ang += 3;
}

//右小腿向上旋转
void inline RotateRightCalfUp(void)
{
    if (G_fRCalf_Ang > G_fRLeg_Ang)
        G_fRCalf_Ang -= 3;
}

//左小腿向下旋转
void inline RotateLeftCalfDown(void)
{
    if (G_fLCalf_Ang < G_fLLeg_Ang + 135)
        G_fLCalf_Ang += 3;
}

//左小腿向上旋转
void inline RotateLeftCalfUp(void)
{
```

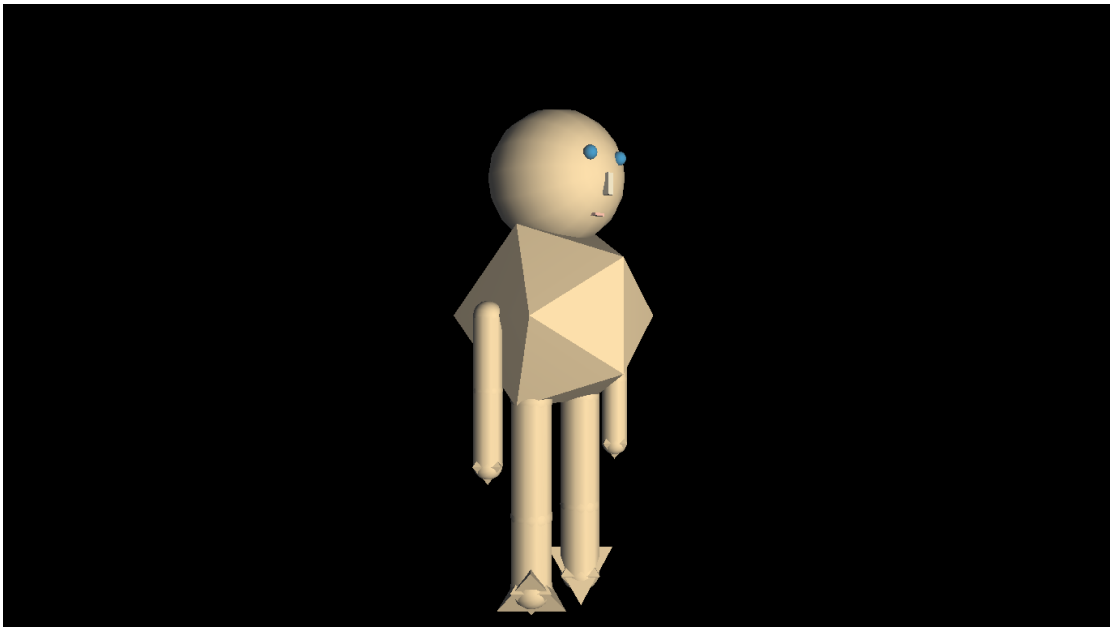
```
{
    if (G_fLCalf_Ang > G_fLLeg_Ang)
        G_fLCalf_Ang -= 3;
}

//重置机器人的状态为初始状态
void inline Reset(void)
{
    G_fDistance = 5.5f;
    G_fDistancce_horizon = 0.0f;
    G_fDistancce_vertical = 0.0f;
    G_fAngle_horizon = 0.0f;
    G_fAngle_vertical = 0.0f;
    G_fRArm_Ang = 0.0f;
    G_fLArm_Ang = 0.0f;
    G_fRLeg_Ang = 0.0f;
    G_fLLeg_Ang = 0.0f;
    G_fRCalf_Ang = 0.0f;
    G_fLCalf_Ang = 0.0f;
    G_fLDArm_Ang = 0.0f;
    G_fRDArm_Ang = 0.0f;
    G_fLArmF_Ang = 0.0f;
    G_fRArmF_Ang = 0.0f;
    G_fHead_Ang = 0.0f;
    G_fRLeg_Horizon = 0.0f;
    G_fLLeg_Horizon = 0.0f;
    G_fLightx = 0.0f;
}

//左下臂向下旋转
void inline RotateLeftDArmDown(void)
{
    if (G_fLArm_Ang - G_fLDArm_Ang < 135)
        G_fLDArm_Ang -= 3;
`
}

//左下臂向下旋转
void inline RotateLeftDArmDown(void)
{
    if (G_fLArm_Ang - G_fLDArm_Ang < 135)
        G_fLDArm_Ang -= 3;
}

//左下臂向上旋转
void inline RotateLeftDArmUp(void) {
    if (G_fLArm_Ang > G_fLDArm_Ang)
        G_fLDArm_Ang += 3;
}

//右下臂向下旋转
void inline RotateRightDArmDown(void)
{
    if (G_fRDArm_Ang - G_fRArm_Ang < 135)
        G_fRDArm_Ang += 3;
}

//右下臂向上旋转
void inline RotateRightDArmUp(void)
{
    if (G_fRArm_Ang < G_fRDArm_Ang)
        G_fRDArm_Ang -= 3;
}

//光照左移
void inline LightMoveLeft(void)
{
    if (G_fLightx > -120)
        G_fLightx -= 3;
`
void inline LightMoveLeft(void)
{
    if (G_fLightx > -120)
        G_fLightx -= 3;
}

//光照右移
void inline LightMoveRight(void)
{
    if (G_fLightx < 120)
        G_fLightx += 3;
}

//左臂向前旋转
void inline RotateLeftArmForward(void)
{
    if (G_fLArmF_Ang > -180)
        G_fLArmF_Ang -= 3;
}

//左臂向后旋转
void inline RotateLeftArmBack(void)
{
    if (G_fLArmF_Ang < 45)
        G_fLArmF_Ang += 3;
}

//右臂向前旋转
void inline RotateRightArmForward(void)
{
    if (G_fRArmF_Ang > -180)
        G_fRArmF_Ang -= 3;
}
```

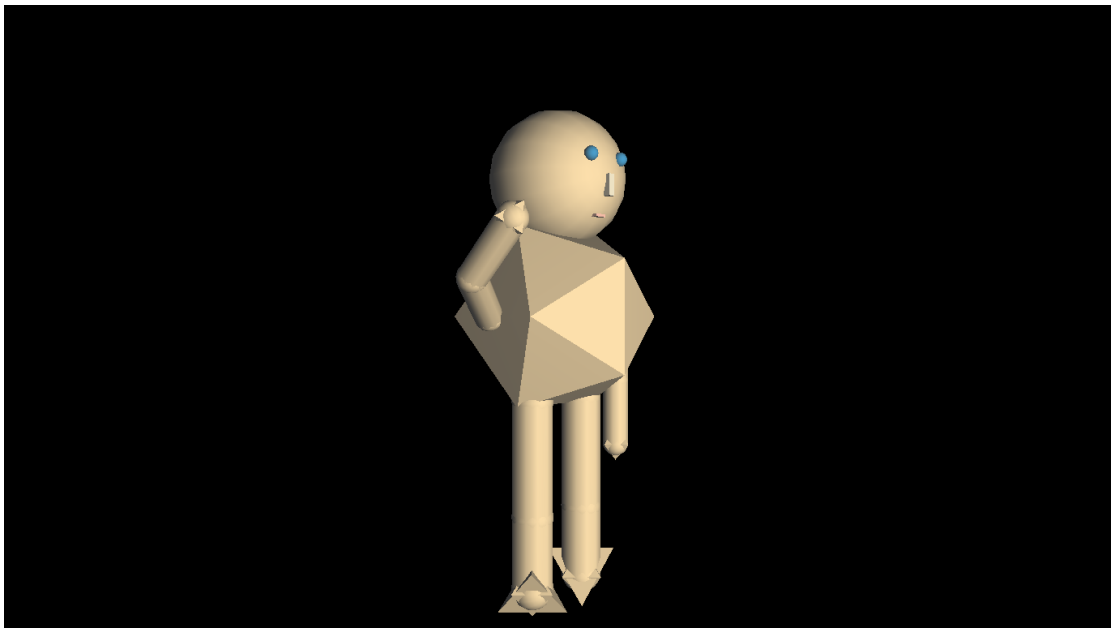
```
        G_fLArmF_Ang += 3;
}

//右臂向前旋转
void inline RotateRightArmForward(void)
{
    if (G_fRArmF_Ang > -180)
        G_fRArmF_Ang -= 3;
}

//右臂向后旋转
void inline RotateRightArmBack(void)
{

    if (G_fRArmF_Ang < 45)
        G_fRArmF_Ang += 3;
}

//脑袋左转
void inline RotateHeadLeft(void)
{
    if (G_fHead_Ang < 90)
        G_fHead_Ang += 3;
}

//脑袋右转
void inline RotateHeadRight(void)
{
    if (G_fHead_Ang > -90)
        G_fHead_Ang -= 3;
}
```

```
                DrawGLScene();                  // Draw The Scene
                SwapBuffers(hDC);               // Swap Buffers (Double Buffering)
            }
        }

        if (keys[VK_F1])                        // Is F1 Being Pressed?
        {
            keys[VK_F1]=false;                  // If So Make Key false
            KillGLWindow();                     // Kill Our Current Window
            fullscreen=!fullscreen;             // Toggle Fullscreen / Windowed Mode
            // Recreate Our OpenGL Window
            if (!CreateGLWindow("机器人",nFullWidth,nFullHeight,32,fullscreen))
            {
                return 0;                       // Quit If Window Was Not Created
            }
        }

        if (keys[VK_F2])
        {
            keys[VK_F2] = false;
            wire = !wire;
        }

        if (keys[VK_SPACE])
        {
            keys[VK_SPACE] = false;
            Reset();
        }

        if(keys[VK_ADD])
        {
            keys[VK_ADD] = false;
            MoveNear();
        }
```

```
        {
            keys[VK_SUBTRACT] = false;
            MoveFar();
        }

        int kg = 0;
        if(keys[VK_INSERT])
            kg += 1;
        if(keys[VK_HOME])
            kg += 2;
        if(keys[VK_DELETE])
            kg += 5;
        if(keys[VK_END])
            kg += 11;
        if(keys[VK_CONTROL])
            kg += 23;
        if(keys[VK_SHIFT])
            kg += 43;
        if(keys[VK_PRIOR])
            kg += 87;
        if(keys[VK_NEXT])
            kg += 173;

        if (keys[VK_LEFT])
        {
            keys[VK_LEFT] = false;
            switch(kg)
            {
            case 0  : LeftRotate();             break;
            case 1  : RotateLeftDArmUp();       break;
            case 2  : RotateRightDArmDown();    break;
            case 5  : RotateLeftCalfDown();     break;
            case 11 : RotateRightCalfDown();    break;
            case 23 : MoveLeft();               break;
```

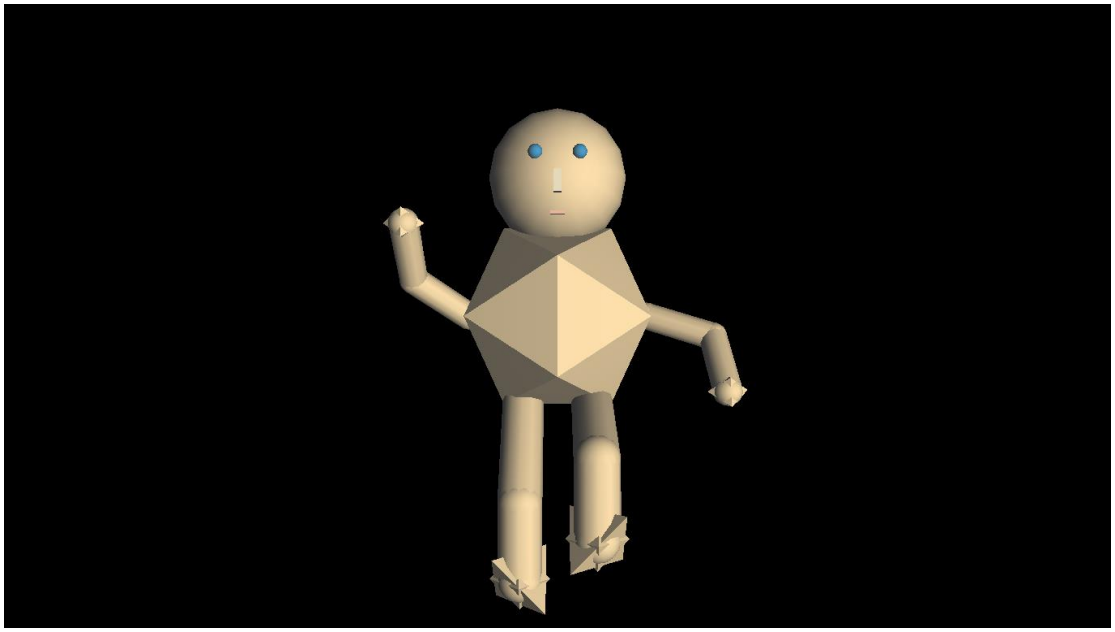
```
                case 2  : RotateRightDArmDown();    break;
                case 5  : RotateLeftCalfDown();     break;
                case 11 : RotateRightCalfDown();    break;
                case 23 : MoveLeft();               break;
                case 43 : LightMoveLeft();          break;
                case 87 : RotateLeftArmForward();   break;
                case 173: RotateLeftArmBack();      break;
                default : break;
                }
            }

            if (keys[VK_RIGHT])
            {
                keys[VK_RIGHT] = false;
                switch(kg)
                {
                case 0  : RightRotate();            break;
                case 1  : RotateLeftDArmDown();     break;
                case 2  : RotateRightDArmUp();      break;
                case 5  : RotateLeftCalfUp();       break;
                case 11 : RotateRightCalfUp();      break;
                case 23 : MoveRight();              break;
                case 43 : LightMoveRight();         break;
                case 87 : RotateRightArmForward();  break;
                case 173: RotateRightArmBack();     break;
                default : break;
                }
            }

            if (keys[VK_UP])
            {
                keys[VK_UP] = false;
                switch(kg)
                {
                case 0  : UpRotate();               break;
                case 1  : RotateLeftArmUp();        break;
                case 2  : RotateRightArmUp();       break;
                case 5  : RotateLeftLegUp();        break;
                case 11 : RotateRightLegUp();       break;
                case 23 : MoveUp();                 break;
                default : break;
                }
            }

            if (keys[VK_DOWN])
            {
                keys[VK_DOWN] = false;
                switch(kg)
                {
                case 0  : DownRotate();             break;
                case 1  : RotateLeftArmDown();      break;
                case 2  : RotateRightArmDown();     break;
                case 5  : RotateLeftLegDown();      break;
                case 11 : RotateRightLegDown();     break;
                case 23 : MoveDown();               break;
                default : break;
                }
            }
        }
    }

    // Shutdown
    KillGLWindow();                                 // Kill The Window
    return (msg.wParam);                            // Exit The Program
}

//移近
void inline MoveNear(void)
```

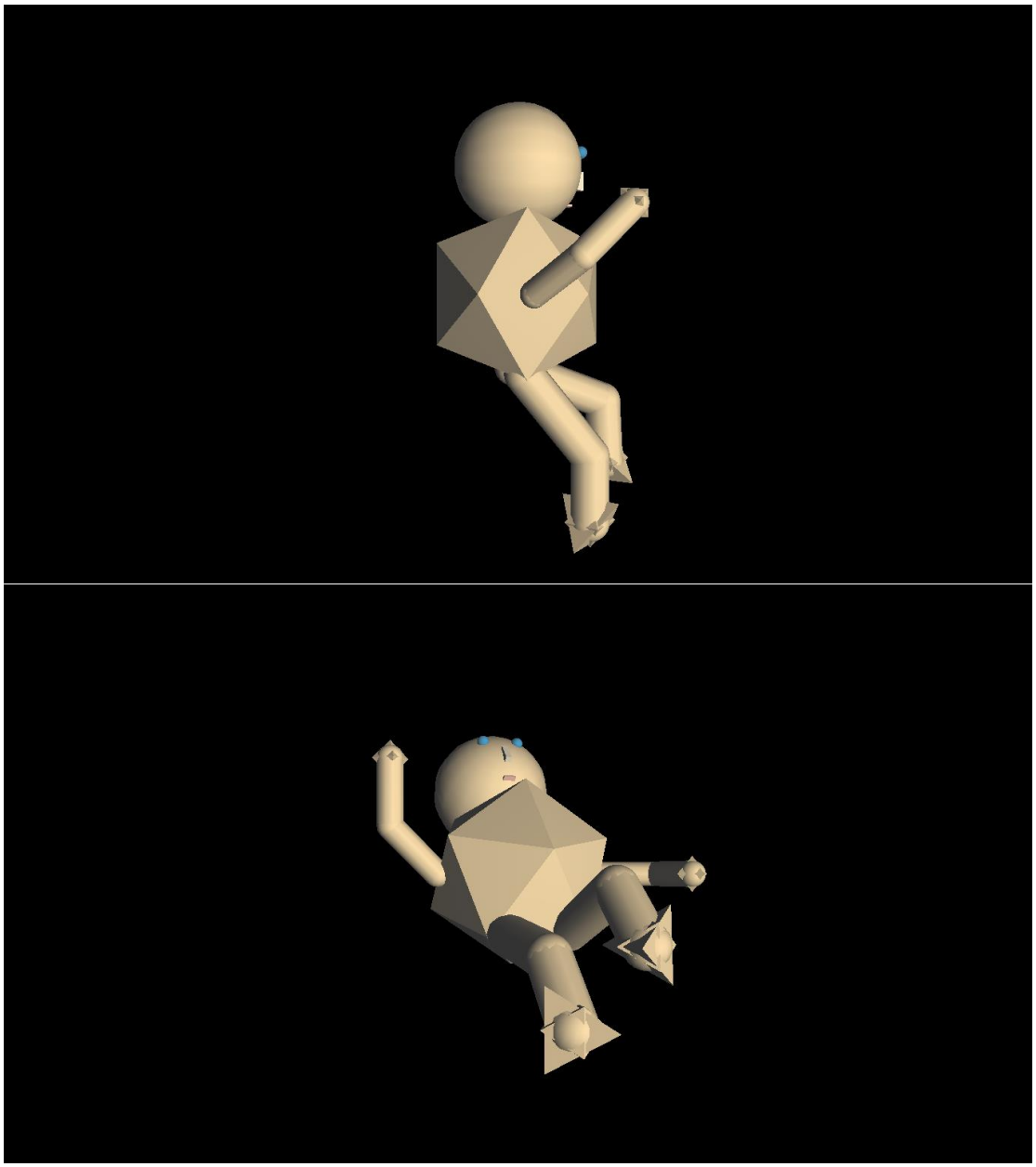


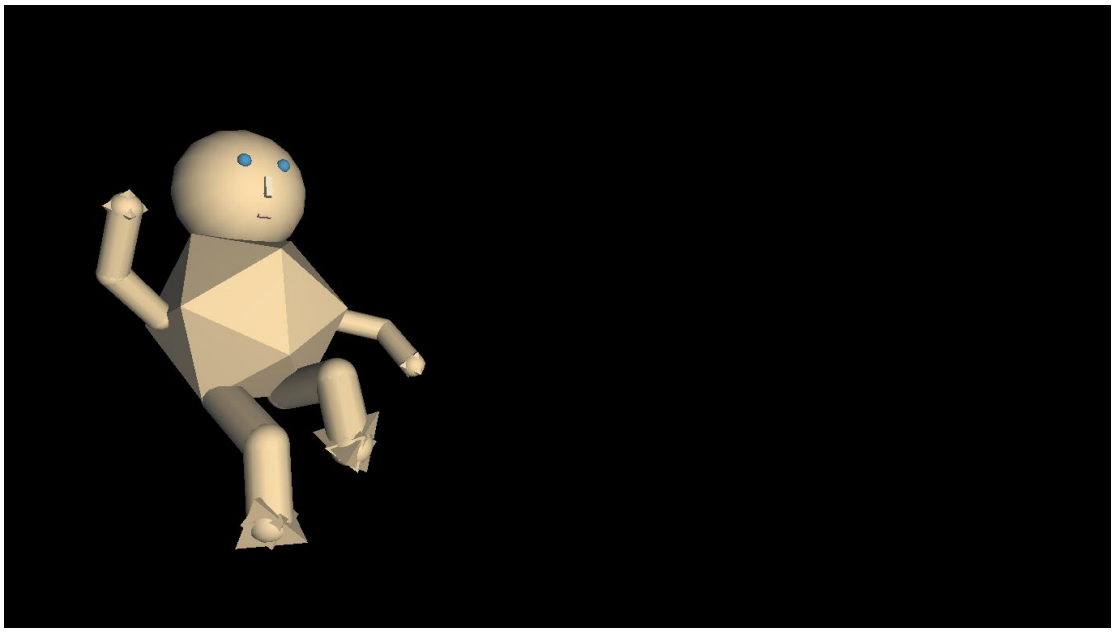


实验结果：

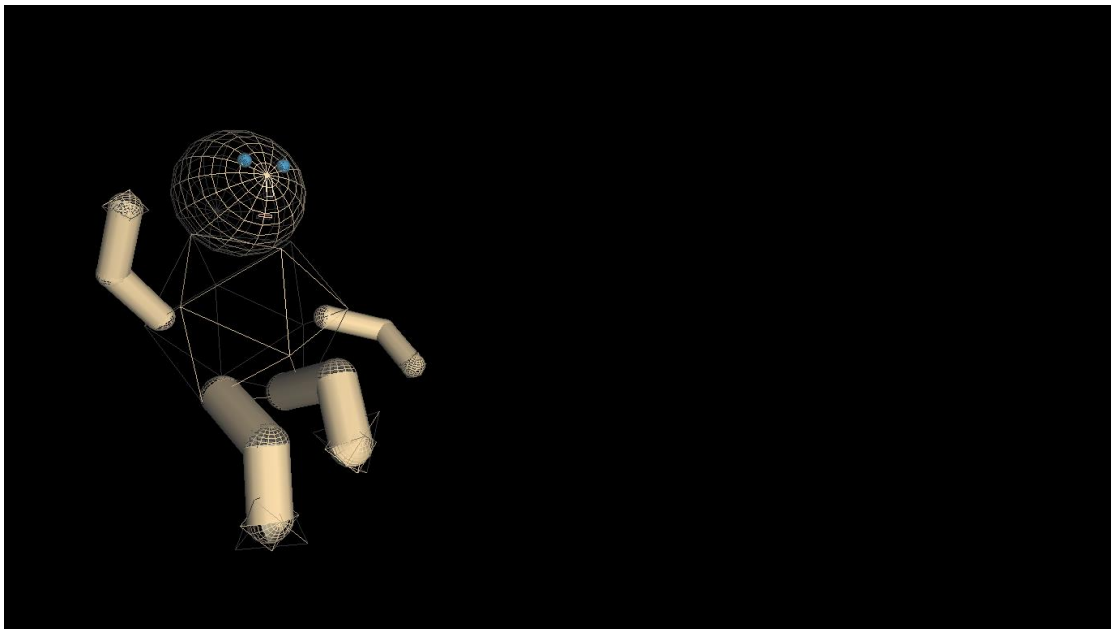
可以灵活转动，并带光照效果的机器人

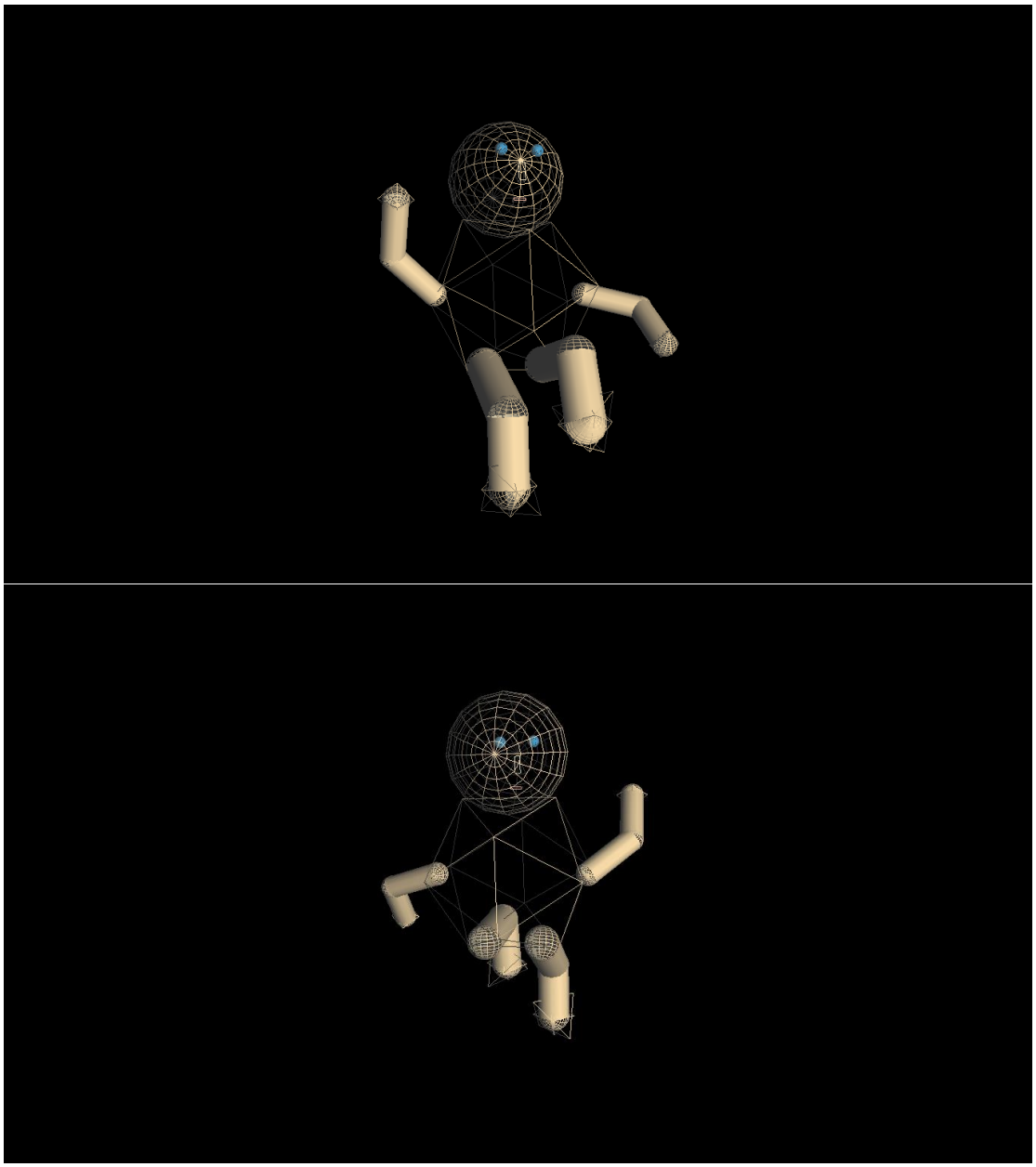




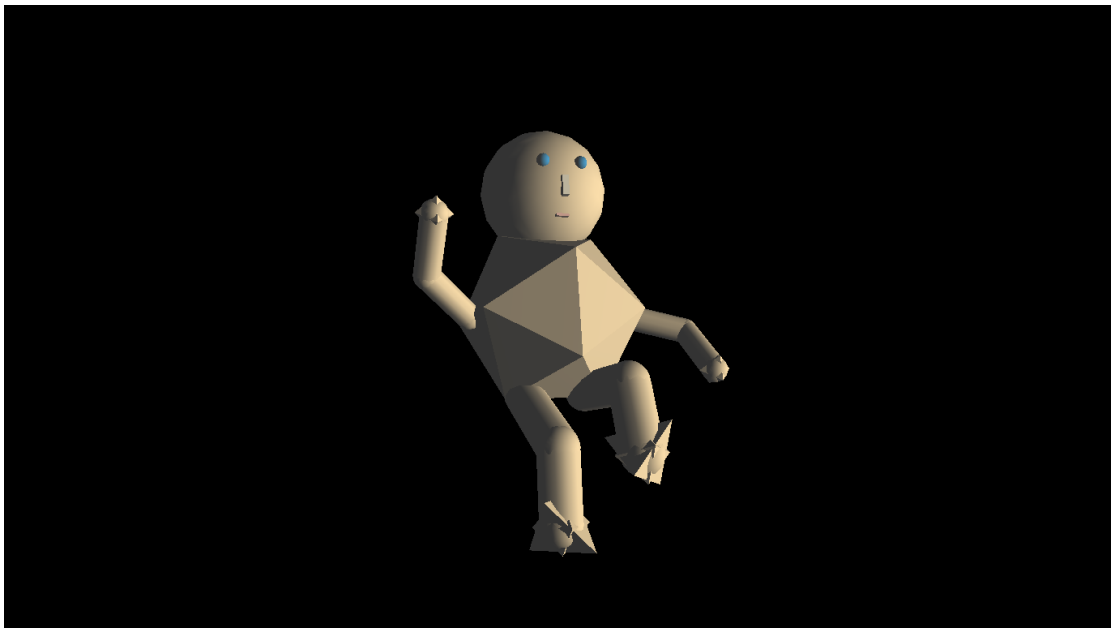


可以切换面片为线框

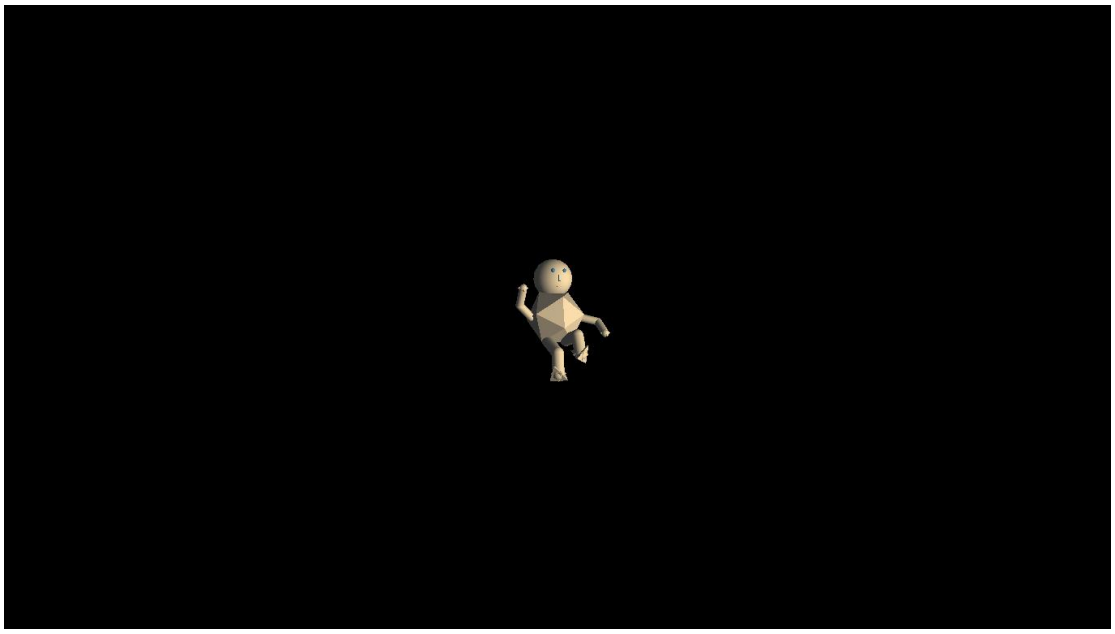


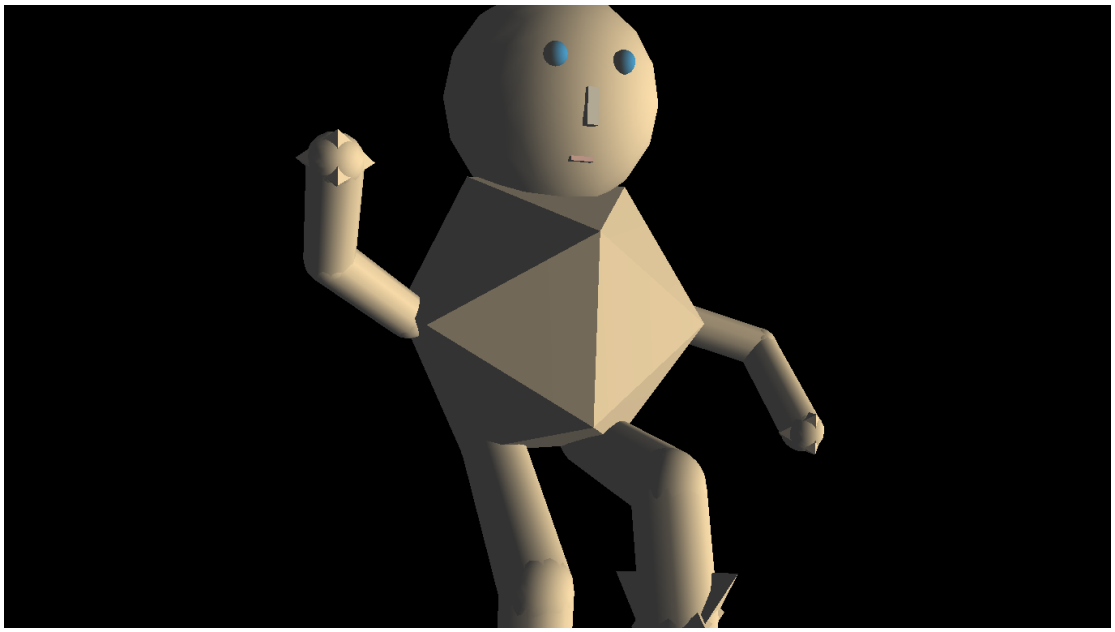


可以改变光照方向

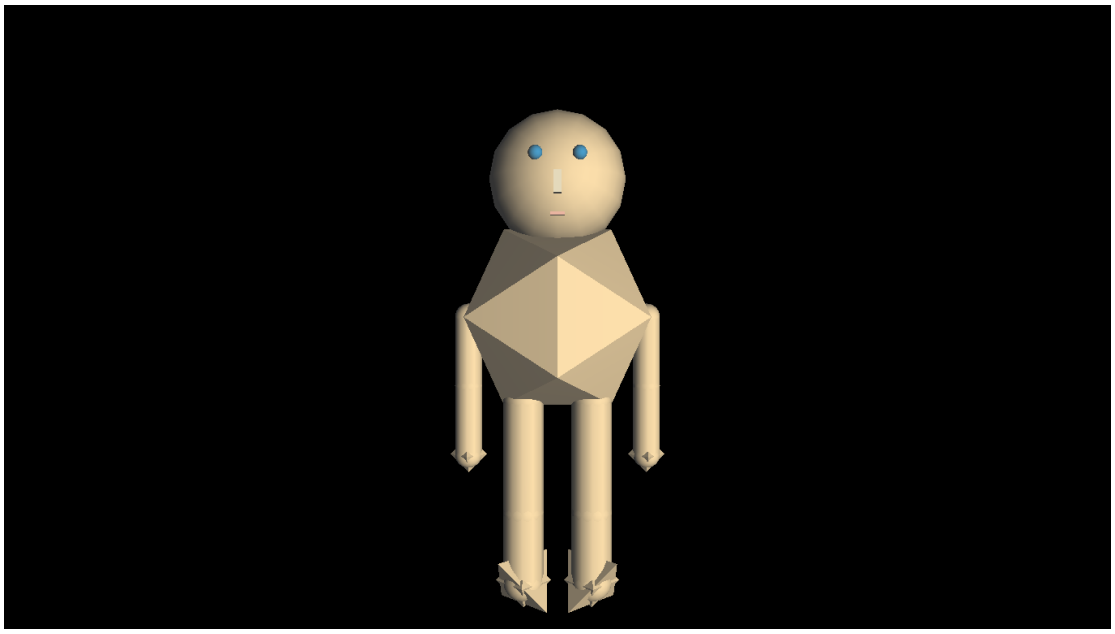


可以拉近拉远

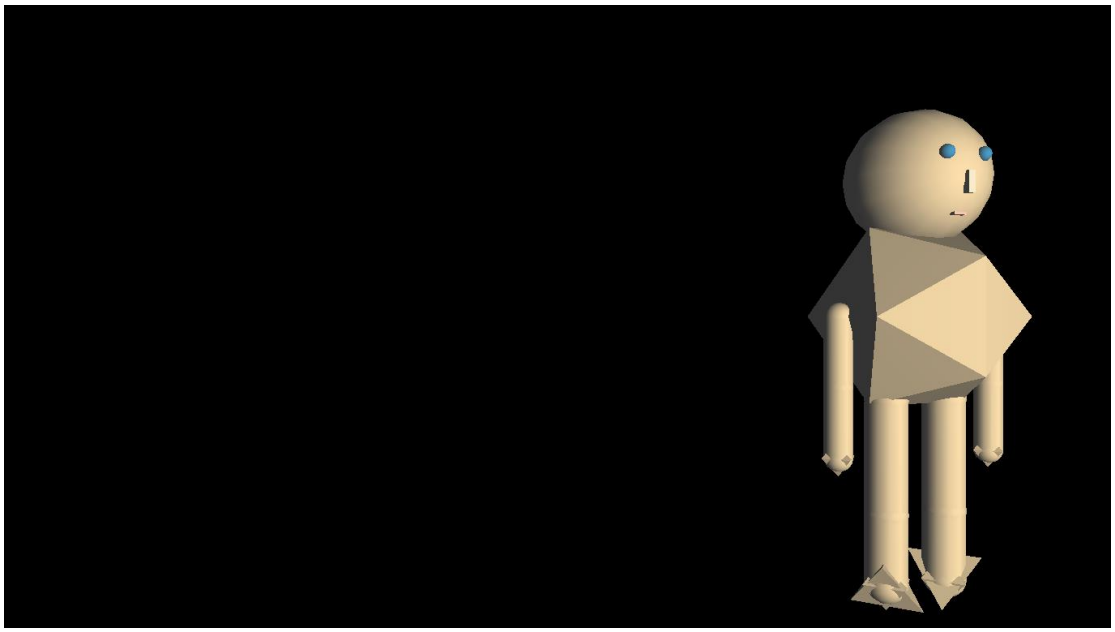




按空格恢复到初始状态



可以上下左右移动



可以旋转

按键说明:

上下左右方向键旋转
Ctrl+上下左右方向键 移动
Shift + 左右方向键 移动光源
Home + 上下左右方向键 移动右臂
End + 上下左右方向键 移动右腿
Insert + 上下左右方向键 移动左臂
Delete + 上下左右方向键 移动左腿
PageUp + 左右方向键 向前移动手臂
PageDown + 左右方向键 向后移动手臂
空格键 重置机器人状态

F1 键 切换全屏与窗口模式
F2 键 切换面片与线框模式
……