コンピューターグラフィックス基礎 第 4 回課題

情報メディア創生学類 3 年 202313625 藤川興昌

実行環境

- Ubuntu 22.04.3 LTS
- gcc version 11.4.0

課題 1

ソースコード

```
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <GL/glut.h>

using namespace std;

// 3�����x�N�g����������ΤN���X
class Vector3d {
public:
        double x, y, z;
        Vector3d() { x = y = z = 0; }
        Vector3d(double _x, double _y, double _z) { x = _x; y = _y; z = _z;
}
        void set(double _x, double _y, double _z) { x = _x; y = _y; z = _z;
}

        // ������1�ɐ��K������
        void normalize() {
                double len = length();
                x /= len; y /= len; z /= len;
        }

        // ������d�
        double length() { return sqrt(x * x + y * y + z * z); }

        // s�{����
        void scale(const double s) { x *= s; y *= s; z *= s; }
```

```
        // ???Z??`
        Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z
+ v.z); }

        // ???Z??`
        // ?????Z??`???Q?l?ʔA???Z?R?[?h???L?q???Ă?
        // ### new program start ###
        Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z
- v.z); }
        // ### new program end ###

        // ???ς?`
        double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }

        // ?O?ς?`
        Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y,
z * v.x - x * v.z, x * v.y - y * v.x); }

        // ???????Z??`
        Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z;
return (*this); }

        // ???Z??????`
        Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z +=
v.z; return (*this); }

        // ???Z??????`
        //
??????Z??????`???Q?l?ʔA???Z????R?[?h???L?q???Ă?
        // ### new program start ###
        Vector3d operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -=
v.z; return (*this); }
        // ### new program end ###

        // ?l???o????
        void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
};

// ?}?C?i?X?'????t??????x?N?g??????????恩?ɔ??邽?Ƥ?` ??Fb=
(-a); ?恩?qL?q?ł???
Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z
) ); }

// ?x?N?g???Ǝ????ς????????恩?ɔ??邽?Ƥ?` ??F c=5*a+2*b; c=b*3;
?恩?qL?q?ł???
Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d(
k*v.x, k*v.y, k*v.z ) );}
Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d(
v.x*k, v.y*k, v.z*k ) );}

// ?x?N?g??????????Ŋ??鍰??????????恩?ɔ??邽?Ƥ?` ??F c=a/2.3;
?恩?qL?q?ł???
Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d(
v.x/k, v.y/k, v.z/k ) );}
```

```
int main(int argc, char**argv) {
        // ### new program start ###
        // x N g Z
        Vector3d a(2, 3, 4);
        Vector3d b(3, 5, -2);
        Vector3d c(2, -1, 1);

        // (a)
        printf("(a) ");
        (a - b).print();

        // (b)
        printf("(b) ");
        Vector3d a3 = a;
        a3.scale(3);
        Vector3d b2 = b;
        b2.scale(2);
        (a3 - b2).print();

        // (c)
        printf("(c) ");
        printf("%lf\n", a * b);

        // (d)
        printf("(d) ");
        Vector3d d = a % b;
        d.normalize();
        d.print();

        // (e)
        printf("(e) ");
        ((a + b2) % c).print();
        // ### new program end ###

        // Visual Studio
        //system("pause");
        return 0;
}
```

結果

- (a) Vector3d(-1.000000 -2.000000 6.000000)
- (b) Vector3d(0.000000 -1.000000 16.000000)
- (c) 13.000000
- (d) Vector3d(-0.851202 0.523816 0.032739)
- (e) Vector3d(13.000000 -8.000000 -34.000000)

課題 2

ソースコード

```
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <GL/glut.h>

using namespace std;

// 3�����x�N�g����������ĤN���X
class Vector3d {
public:
        double x, y, z;
        Vector3d() { x = y = z = 0; }
        Vector3d(double _x, double _y, double _z) { x = _x; y = _y; z = _z;
}
        void set(double _x, double _y, double _z) { x = _x; y = _y; z = _z;
}

        // ������1�e��K������
        void normalize() {
                double len = length();
                x /= len; y /= len; z /= len;
        }

        // ������d�
        double length() { return sqrt(x * x + y * y + z * z); }

        // s�{����
        void scale(const double s) { x *= s; y *= s; z *= s; }

        // ���Z��`
        Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z
+ v.z); }

        // ���Z��`
        // ������Z��`���Q�l�?A���Z�R�[�h���L�q���Ă�
        // ### new program start ###
        Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z
- v.z); }
        // ### new program end ###

        // ���ς̀
        double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }

        // ⊗O̲ς̀
        Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y,
z * v.x - x * v.z, x * v.y - y * v.x); }

        // ������Z��`
```

```
        Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z;
return (*this); }

        // ???Z?????`
        Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z +=
v.z; return (*this); }


        // ???Z?????`
        //
?????Z?????`???Q?l?A???Z????R?[?h???L?q???Ă?
        // ### new program start ###
        Vector3d operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -=
v.z; return (*this); }
        // ### new program end ###

        // ?l???o????
        void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
};

// ?}?C?i?X?'????t?????x?N?g??????????恩????邽?Ƥ?` ??Fb=
(-a); ?恩?qL?q?ł???
Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z
) ); }

// ?x?N?g???Ǝ????ς????????恩????邽?Ƥ?` ??F c=5*a+2*b; c=b*3;
?恩?qL?q?ł???
Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d(
k*v.x, k*v.y, k*v.z ) );}
Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d(
v.x*k, v.y*k, v.z*k ) );}

// ?x?N?g??????????D??鑚??????????恩????邽?Ƥ?` ??F c=a/2.3;
?恩?qL?q?ł???
Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d(
v.x/k, v.y/k, v.z/k ) );}


// ??̃?????i?[????N???X
class Sphere {
public:
        Vector3d position; // ???S?ʒu
        float color[3];    // ?`??F

        void setColor(float r, float g, float b) {
                color[0] = r; color[1] = g; color[2] = b;
        }

        // ???????`?惊?郁???o?▫?
        void display() {
                glPushMatrix(); // ????̊f???ï??s???ᄀ????Ă???

                // ???W?'??s?ᵩ??ƀX?P?[???ï????{???ċ???`?惊??
                glTranslated(position.x, position.y, position.z);
                glScaled(2, 2, 2);
                glutSolidSphere(1.0, 32, 32);
```

```
                glPopMatrix();  // ﾂ・ﾂ・ﾂ・
```

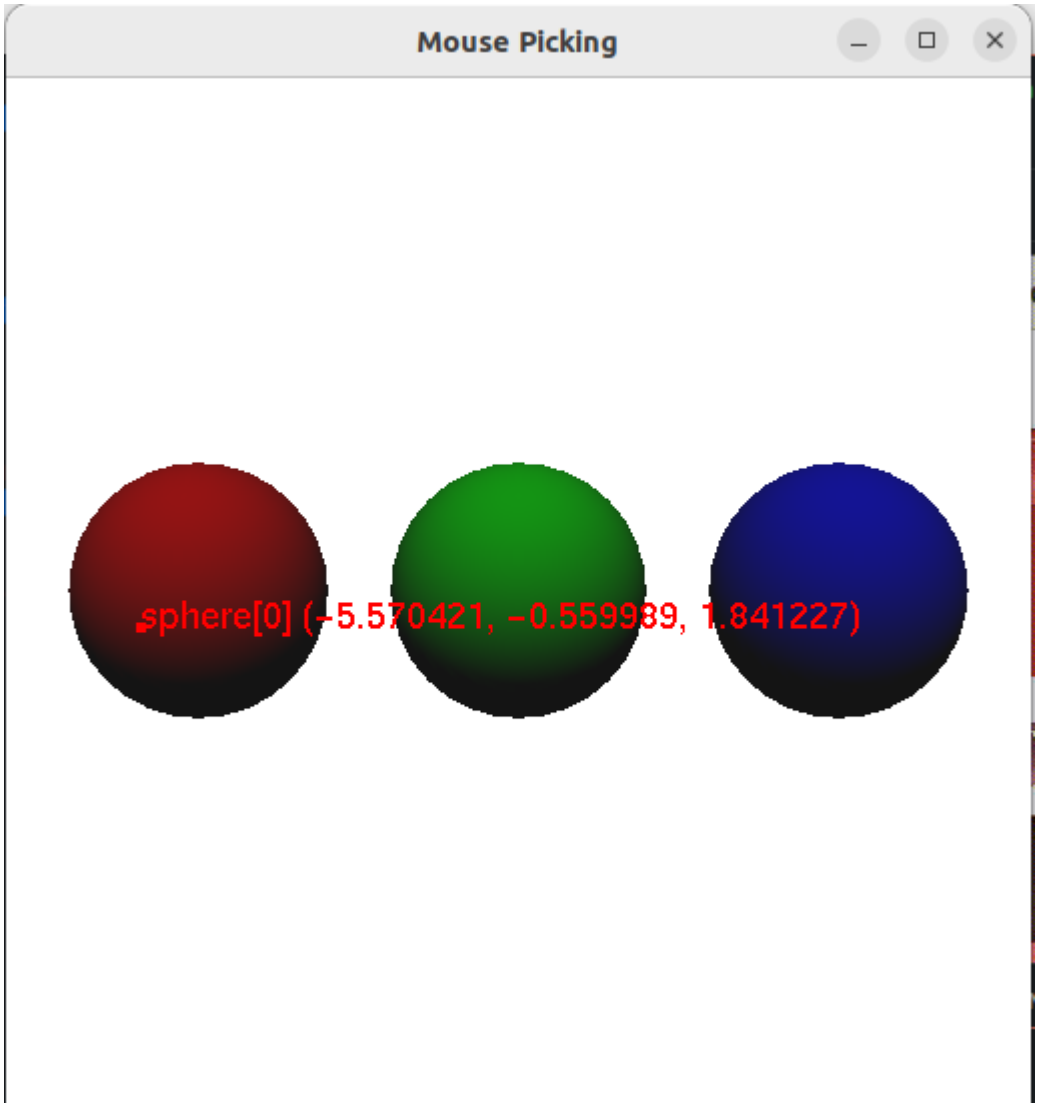
```
100.0);

        // J[W
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(0, 0, 30, 0, 0, 0, 0, 1, 0);

        // 3
        for (int i = 0; i < 3; i++) {
                // F
                glMaterialfv(GL_FRONT, GL_DIFFUSE, g_Sphere[i].color);

                //
                g_Sphere[i].display();
        }

        //
I
A
        if(g_SelectedSphereID != -1) {
                // l
                glDisable(GL_LIGHTING);
                glDisable(GL_DEPTH_TEST);

                // N
                glColor3f(1, 0, 0);
                glPointSize(5.f);
                glBegin(GL_POINTS);
                glVertex3d(g_SelectedPos.x, g_SelectedPos.y,
g_SelectedPos.z);
                glEnd();

                //
                glRasterPos3d(g_SelectedPos.x, g_SelectedPos.y,
g_SelectedPos.z);

                // \z
                //  sprintf_s R
sprintf g
                char str[256];
                sprintf(str, "sphere[%d] (%lf, %lf, %lf)",
g_SelectedSphereID,
                                g_SelectedPos.x, g_SelectedPos.y,
g_SelectedPos.z);

                // 1
                for (int i = 0; str[i] != '\0'; i++) {
                        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,
str[i]);
                }
        }

        glutSwapBuffers();
}

// EBhETCYX
void resize(int w, int h) {
```

```
        if (h < 1) return;

        glViewport(0, 0, w, h);

        g_WindowWidth = w;
        g_WindowHeight = h;
}

// }E X J[ \ ƷuQ I
void MousePick(int x, int _y) {

        printf("MousePick(%d, %d)\n", x, _y);

        // }E X N b N W _ 
OpenGL W n  ] 
        const int y = g_WindowHeight - _y;

        g_SelectedSphereID = pickSphere(x, y);

        // I Ă  
        if (g_SelectedSphereID == -1) return;

        // N b N W l i3 W j 

        // X C h Q l Y 
 v O R [ h lj 
        //  f r [ s 
        //  e s  
        //  [ | [ g 
        // } E X N b N ƷuQ s iz l j 
 
        //
 L Q  ĂA N b N ƷuQ [ h W 
        //  l Ag_SelectedPos
 Q i [ Ă i \ Yg p j
        // ### new program start ###
        double M[16];  // モデルビュー行列の取得
        glGetDoublev(GL_MODELVIEW_MATRIX, M);
        double P[16];  // 透視投影行列の取得
        glGetDoublev(GL_PROJECTION_MATRIX, P);
        int V[4];  // ビューポートの情報を取得
        glGetIntegerv(GL_VIEWPORT, V);
        float z;
        glReadPixels(x, y, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &z);
        gluUnProject(x, y, z, M, P, V, &g_SelectedPos.x, &g_SelectedPos.y,
&g_SelectedPos.z);
        // ### new program end ###
}

// }E X N b N C x g 
void mouse(int button, int state, int x, int y) {
        if (state == GLUT_DOWN) MousePick(x, y);
        glutPostRedisplay();
}

// }E X h b O C x g 
void motion(int x, int y) {
```

```
        MousePick(x, y);
        glutPostRedisplay();
}

// ｷLｷ[ｷｷｷｷｷｷｷ｡｡ＢｷｷCｷxｷｷｷgｷｷｷｷ
void keyboard(unsigned char key, int x, int y) {
        switch (key) {
        case 'q':
        case 'Q':
        case '\033':
                exit(0);  /* '\033' ｷｷ ESC ｷｷ ASCII ｷRｷ[ｷh */
        default:
                break;
        }

        glutPostRedisplay();
}


void init() {
        // 3ｷ｡ｷ｡ｷｷｷｚuｷＥＦｷｷ･脈ｷＡｷｷｷ
        g_Sphere[0].position.set(-5, 0, 0);
        g_Sphere[1].position.set( 0, 0, 0);
        g_Sphere[2].position.set( 5, 0, 0);
        g_Sphere[0].setColor(1, 0, 0);
        g_Sphere[1].setColor(0, 1, 0);
        g_Sphere[2].setColor(0, 0, 1);

        glClearDepth(1000.0);
        glClearColor(1, 1, 1, 1); // ｷwｷiｷＦｷ｡ｵ･ｷ

        // ｷlｷｷ･ｷ
        float lightAmbientColor[] = { 0.2f, 0.2f, 0.2f, 0.0f };
        float lightDiffuseColor[] = { 1.f, 1.f, 1.f, 0.0f };
        float lightSpecularColor[] = { 0.4f, 0.4f, 0.4f, 0.0f };
        float lightPosition[] = { 0.0f, 30.0f, 30.0f, 0.0f };
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbientColor);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuseColor);
        glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecularColor);
        glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

        // ｷ･ｷｷ､ｷ
        float specularColor[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        float ambientColor[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        float diffuseColor[] = { 1.f, 0.f, 0.f, 1.f };
        float shininess = 64.f;
        glMaterialfv(GL_FRONT, GL_SPECULAR, specularColor);
        glMaterialfv(GL_FRONT, GL_SHININESS, &shininess);
        glMaterialfv(GL_FRONT, GL_AMBIENT, ambientColor);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuseColor);
}

int main(int argc, char**argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
```

```
        glutInitWindowSize(g_WindowWidth, g_WindowHeight);
        glutCreateWindow("Mouse Picking");

        glutDisplayFunc(display);
        glutReshapeFunc(resize);
        glutMouseFunc(mouse);
        glutMotionFunc(motion);
        glutKeyboardFunc(keyboard);

        init();

        glutMainLoop();

        return 0;
}
```

スクリーンショット



発展課題

ソースコード

```
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <GL/glut.h>

using namespace std;

// 3�����x�N�g����������ΨN���X
class Vector3d {
public:
        double x, y, z;
        Vector3d() { x = y = z = 0; }
        Vector3d(double _x, double _y, double _z) { x = _x; y = _y; z = _z;
}
        void set(double _x, double _y, double _z) { x = _x; y = _y; z = _z;
}

        // ������1�ɐ��K������
        void normalize() {
                double len = length();
                x /= len; y /= len; z /= len;
        }

        // ������d�
        double length() { return sqrt(x * x + y * y + z * z); }

        // s�{����
        void scale(const double s) { x *= s; y *= s; z *= s; }

        // ���Z��`
        Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z
+ v.z); }

        // ���Z��`
        // ����Z��`���Q�l�ΏA���Z�R�[�h���L�q���Ă�
        Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z
- v.z); }

        // ���ς̉`
        double operator*(Vector3d v) { return x * v.x + y* v.y + z * v.z; }

        // ΋O̓ς̉`
        Vector3d operator%(Vector3d v) { return Vector3d(y * v.z - z * v.y,
z * v.x - x * v.z, x * v.y - y * v.x); }

        // ������Z��`
        Vector3d& operator=(const Vector3d& v){ x = v.x; y = v.y; z = v.z;
return (*this); }

        // ���Z�����`
        Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z +=
v.z; return (*this); }
```

```
        // ???Z?????`
        //
??????Z??????`???Q?l?A???Z????R?[?h???L?q???Ă?
        Vector3d operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -=
v.z; return (*this); }

        // ?l???o????
        void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }
};

// ?}?C?i?X?'????t??????x?N?g???????????恩????邯?Ƥ?` ??Fb=
(-a); ?恩?qL?q?ł???
Vector3d operator-( const Vector3d& v ) { return( Vector3d( -v.x, -v.y, -v.z
) ); }

// ?x?N?g???Ǝ????ς????????恩????邯?Ƥ?` ??F c=5*a+2*b; c=b*3;
?恩?qL?q?ł???
Vector3d operator*( const double& k, const Vector3d& v ) { return( Vector3d(
k*v.x, k*v.y, k*v.z ) );}
Vector3d operator*( const Vector3d& v, const double& k ) { return( Vector3d(
v.x*k, v.y*k, v.z*k ) );}

// ?x?N?g??????????D??鑁??????????恩????邯?Ƥ?` ??F c=a/2.3;
?恩?qL?q?ł???
Vector3d operator/( const Vector3d& v, const double& k ) { return( Vector3d(
v.x/k, v.y/k, v.z/k ) );}


// ??̈́?????i?[????N???X
class Sphere {
public:
        Vector3d position; // ???S?ʒu
        float color[3];    // ?`??F
        int score;

        void setColor(float r, float g, float b) {
                color[0] = r; color[1] = g; color[2] = b;
        }

        // ???????`?惊?郁???o?▫?
        void display() {
                glPushMatrix(); // ????̣??f???ï??s???ヶ????Ă???

                // ???W???s?ᵩ??ƀX?P?[???ï????{???ċ???`?惊??
                glTranslated(position.x, position.y, position.z);
                glScaled(2, 2, 2);
                glutSolidSphere(1.0, 32, 32);

                glPopMatrix();  // ?ヶ????Ă???????f???ï??s???Ƥ?
        }
};

// 3?▫?̋??????????Ă???
Sphere g_Sphere[3];

// ?I????ძ???鋅
```

```
��ID�G��i0,1,2�j��ぢ�����B�I����ძ��������-1�Б���B
int g_SelectedSphereID = -1;

// �N���b�N���▫3▫▫▫▫▫▫W��ぢ�����
Vector3d g_SelectedPos;

const int g_AnimationIntervalMsec = 10;
int g_RotatedAngle = 0;
const double g_CameraRotatedR = 40.0;
int g_CameraRotatedTheta = 0;
int g_CameraRotatedPhi = 0;
int g_Score = 0;

// �E�B���h�E�T�C�Y��ぢ�����
int g_WindowWidth = 512;
int g_WindowHeight = 512;

// �I�������ID�G��i0,1,2�j��ძ
// �I������������▫▫▫ -1 ▫▫ძ
int pickSphere(int x, int y) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable(GL_DEPTH_TEST);

        // �l����Ş▫▫▫▫ĒP�F�ŕ`�惊��
        glDisable(GL_LIGHTING);

        // 3�▫ű���`�惊��
        for (int i = 0; i < 3; i++) {
                // RGB��R�����q���ID��ټ欹��(unsigned byte�^)
                glColor3ub(i, 0, 0);
                g_Sphere[i].display();
        }

        // ����ƀX���C�h���Q�l�Ɏ��恩
�âv���O�����R�[�h��lj�����
        // ��glReadPixels
�▫����(����āA�N���b�N�����ʒu�����F�ł��烳擾����
        // ���擾�����F�����āA�ǂ��I�������斫
�A����ID�� return ����B
        GLubyte c[3];
        glReadPixels(x, y, 1, 1, GL_RGB, GL_UNSIGNED_BYTE, c);

        if(0 <= c[0] && c[0] <= 3) g_Score += g_Sphere[c[0]].score;
        return c[0] <= 3 ? c[0] : -1; // ���K�ۣR̂l��ძ��恩�ヮ���
}

// ��`��▫�
void display() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);

        // ��������e�ï��s��ټ
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(30.0, g_WindowWidth / (float)g_WindowHeight, 1.0,
```

```
100.0);

        // J[W
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(g_CameraRotatedR*cos(g_CameraRotatedPhi /
180.0)*cos(g_CameraRotatedTheta / 180.0),
g_CameraRotatedR*sin(g_CameraRotatedPhi / 180.0),
g_CameraRotatedR*cos(g_CameraRotatedPhi / 180.0)*sin(g_CameraRotatedTheta /
180.0), 0, 0, 0, 0, 1, 0);

        g_Sphere[0].position.set(10 * sin(M_PI * g_RotatedAngle*2 / 180.0),
10 * cos(M_PI * g_RotatedAngle*2 / 180.0), 0);
        g_Sphere[1].position.set(10 * sin(M_PI * g_RotatedAngle / 180.0), 0,
10 * cos(M_PI * g_RotatedAngle*3 / 180.0));
        g_Sphere[2].position.set(0, 10 * sin(M_PI * g_RotatedAngle*3 /
180.0), 10 * cos(M_PI * g_RotatedAngle / 180.0));

        // 3
        for (int i = 0; i < 3; i++) {
                // F
                glMaterialfv(GL_FRONT, GL_DIFFUSE, g_Sphere[i].color);

                //
                g_Sphere[i].display();
        }

        //
I
        if(g_SelectedSphereID != -1) {
                // l
                glDisable(GL_LIGHTING);
                glDisable(GL_DEPTH_TEST);

                // N
                glColor3f(1, 0, 0);
                glPointSize(5.f);
                glBegin(GL_POINTS);
                glVertex3d(g_SelectedPos.x, g_SelectedPos.y,
g_SelectedPos.z);
                glEnd();

                //
                glRasterPos3d(g_SelectedPos.x, g_SelectedPos.y,
g_SelectedPos.z);

                // \
                // sprintf_s R sprintf g
                char str[256];
                sprintf(str, "sphere[%d] +%d", g_SelectedSphereID,
g_Sphere[g_SelectedSphereID].score);

                // 1
                for (int i = 0; str[i] != '\0'; i++) {
                        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,
```

```
str[i]);
                }
        }

        Vector3d label_pos;
        double M[16];  // モデルビュー行列の取得
        glGetDoublev(GL_MODELVIEW_MATRIX, M);
        double P[16];  // 透視投影行列の取得
        glGetDoublev(GL_PROJECTION_MATRIX, P);
        int V[4];  // ビューポートの情報を取得
        glGetIntegerv(GL_VIEWPORT, V);
        float z;
        glReadPixels(25, 25, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &z);
        gluUnProject(25, 25, z, M, P, V, &label_pos.x, &label_pos.y,
&label_pos.z);
        glRasterPos3d(label_pos.x, label_pos.y, label_pos.z);
        // ?\??????????????\?z
        // ??????? sprintf_s ?N?R???p?C???G????[?2?????? sprintf
???g???????
        char label_str[256];
        sprintf(label_str, "Score: %d", g_Score);
        // ????????1????????`??
        for (int i = 0; label_str[i] != '\0'; i++) {
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, label_str[i]);
        }

        glutSwapBuffers();
}

// ?E?B???h?E?T?C?Y???X????????B??????
void resize(int w, int h) {
        if (h < 1) return;

        glViewport(0, 0, w, h);

        g_WindowWidth = w;
        g_WindowHeight = h;
}

// ?}?E?X?J?[?\???3u?Q?A??I???????
void MousePick(int x, int _y) {

        printf("MousePick(%d, %d)\n", x, _y);

        // ?}?E?X?N???b?N?œ?????????W???????_?A?
OpenGL???W?n?Q??킷?邳?H?y???W?]????
        const int y = g_WindowHeight - _y;

        g_SelectedSphereID = pickSphere(x, y);

        // ?????I???????A??A??A牽?????A?
        if (g_SelectedSphereID == -1) return;

        // ?N???b?N??????????W?l?i3?????????W?j???擾????

        // ?????BX???C?h???Q?l?Y??恩
?av???O??????R?[?h??lj??????
```

```
        // ������f���r���[s����擾����
        // ������e s擾���擾����
        // ����r���[|[g�����擾
        // ���}E X N���b N�����ʒu���s�����iz l j���擾
����
        //
����L���Q Â āA N���b N�����ʒu���[���h���W���擾����
        // ���擾�����l Ag_SelectedPos
Qi[���Ă����i���\�����Ygp����j
        double M[16];  // モデルビュー行列の取得
        glGetDoublev(GL_MODELVIEW_MATRIX, M);
        double P[16];  // 透視投影行列の取得
        glGetDoublev(GL_PROJECTION_MATRIX, P);
        int V[4];  // ビューポートの情報を取得
        glGetIntegerv(GL_VIEWPORT, V);
        float z;
        glReadPixels(x, y, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &z);
        gluUnProject(y, x, z, M, P, V, &g_SelectedPos.x, &g_SelectedPos.y,
&g_SelectedPos.z);
}

// }E X N���b N C x���g����
void mouse(int button, int state, int x, int y) {
        if (state == GLUT_DOWN) MousePick(x, y);
        glutPostRedisplay();
}

// }E X h���b O C x���g����
void motion(int x, int y) {
        MousePick(x, y);
        glutPostRedisplay();
}

// L[������▫▫Б��C x���g����
void keyboard(unsigned char key, int x, int y) {
        switch (key) {
        case 'q':
        case 'Q':
        case '\033':
                exit(0);  /* '\033' ?? ESC ?? ASCII R[h */
        case 'w':
        case 'W':
                g_CameraRotatedPhi -= 10;
                break;
        case 'a':
        case 'A':
                g_CameraRotatedTheta -= 10;
                break;
        case 's':
        case 'S':
                g_CameraRotatedPhi += 10;
                break;
        case 'd':
        case 'D':
                g_CameraRotatedTheta += 10;
                break;
        default:
```

```
                break;
        }

        glutPostRedisplay();
}


void timer(int val) {
        g_RotatedAngle++;

        glutPostRedisplay();
        glutTimerFunc(g_AnimationIntervalMsec, timer, val);
}


void init() {
        // 3□ŕ□□ʒu□ƐF□□ﾟ脈□Ă□□□
        g_Sphere[0].position.set(0, 0, 0);
        g_Sphere[1].position.set(0, 0, 0);
        g_Sphere[2].position.set(0, 0, 0);
        g_Sphere[0].setColor(1, 0, 0);
        g_Sphere[1].setColor(0, 1, 0);
        g_Sphere[2].setColor(0, 0, 1);
        g_Sphere[0].score = 1;
        g_Sphere[1].score = 10;
        g_Sphere[2].score = 100;

        glClearDepth(1000.0);
        glClearColor(1, 1, 1, 1); // □w□i□F□□ﾟ□

        // □l□□ﾟ□
        float lightAmbientColor[] = { 0.2f, 0.2f, 0.2f, 0.0f };
        float lightDiffuseColor[] = { 1.f, 1.f, 1.f, 0.0f };
        float lightSpecularColor[] = { 0.4f, 0.4f, 0.4f, 0.0f };
        float lightPosition[] = { 0.0f, 30.0f, 30.0f, 0.0f };
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbientColor);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuseColor);
        glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecularColor);
        glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

        // □ﾟ□□ｰ□
        float specularColor[] = { 0.8f, 0.8f, 0.8f, 1.0f };
        float ambientColor[] = { 0.2f, 0.2f, 0.2f, 1.0f };
        float diffuseColor[] = { 1.f, 0.f, 0.f, 1.f };
        float shininess = 64.f;
        glMaterialfv(GL_FRONT, GL_SPECULAR, specularColor);
        glMaterialfv(GL_FRONT, GL_SHININESS, &shininess);
        glMaterialfv(GL_FRONT, GL_AMBIENT, ambientColor);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuseColor);
}

int main(int argc, char**argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowSize(g_WindowWidth, g_WindowHeight);
```

```
        glutCreateWindow("Mouse Picking");

        glutDisplayFunc(display);
        glutReshapeFunc(resize);
        glutMouseFunc(mouse);
        glutMotionFunc(motion);
        glutKeyboardFunc(keyboard);
        glutTimerFunc(g_AnimationIntervalMsec, timer, 0);

        init();

        glutMainLoop();

        return 0;
}
```

スクリーンショット