CG基礎 第四回レポート

201811395 山本雄太

課題4(1)のソースコード

|  |
| --- |
| #define FREEGLUT\_STATIC  #include <cstdlib>  #include <cstdio>  #include <cmath>  #include <GL/glut.h>  using namespace std;  // 3次元ベクトルを扱うためのクラス  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; }  // 長さを1に正規化する  void normalize() {  double len = length();  x /= len; y /= len; z /= len;  }  // 長さを返す  double length() { return sqrt(x \* x + y \* y + z \* z); }  // s倍する  void scale(const double s) { x \*= s; y \*= s; z \*= s; }  // 加算の定義  Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z + v.z); }  // 減算の定義  ***Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z - v.z); }***  // 内積の定義  double operator\*(Vector3d v) { return x \* v.x + y \* v.y + z \* v.z; }  // 外積の定義  Vector3d operator%(Vector3d v) { return Vector3d(y \* v.z - z \* v.y, z \* v.x - x \* v.z, x \* v.y - y \* v.x); }  // 代入演算の定義  Vector3d& operator=(const Vector3d& v) { x = v.x; y = v.y; z = v.z; return (\*this); }  // 加算代入の定義  Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z += v.z; return (\*this); }  // 減算代入の定義  ***Vector3d& operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -= v.z; return (\*this); }***  // 値を出力する  void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }  };  // マイナスの符号の付いたベクトルを扱えるようにするための定義 例：b=(-a); のように記述できる  Vector3d operator-(const Vector3d& v) { return(Vector3d(-v.x, -v.y, -v.z)); }  // ベクトルと実数の積を扱えるようにするための定義 例： c=5\*a+2\*b; c=b\*3; のように記述できる  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)); }  // ベクトルを実数で割る操作を扱えるようにするための定義 例： c=a/2.3; のように記述できる  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) {  // ベクトルの加算の例  ***Vector3d a(2, 3, 4);***  ***Vector3d b(3, 5, -2);***  ***Vector3d c(2, -1, 1);***  ***Vector3d d = a - b;***  ***Vector3d e = 3\*a - 2\*b;***  ***double f = a \* b;***  ***Vector3d g = a % b;***  ***Vector3d h = (a + 2 \* b) % c;***  ***d.print();***  ***e.print();***  ***printf("%lf\n", f);***  ***g.print();***  ***h.print();***  // Visual Studio でコンソールがすぐに閉じないようにするためには次のコメントアウトをはずす  //system("pause");  return 0;  } |

サンプルコードに書き足した部分を赤くした。

これを実行することにより得られた問題の答えは以下の通り。

1. Vector3d(-1.000000 -2.000000 6.000000)
2. Vector3d(0.000000 -1.000000 16.000000)
3. 13.000000
4. Vector3d(-26.000000 16.000000 1.000000)
5. Vector3d(13.000000 -8.000000 -34.000000)

課題4(2)のソースコード

|  |
| --- |
| #define FREEGLUT\_STATIC  #include <cstdlib>  #include <cstdio>  #include <cmath>  #include <GL/glut.h>  using namespace std;  // 3次元ベクトルを扱うためのクラス  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; }  // 長さを1に正規化する  void normalize() {  double len = length();  x /= len; y /= len; z /= len;  }  // 長さを返す  double length() { return sqrt(x \* x + y \* y + z \* z); }  // s倍する  void scale(const double s) { x \*= s; y \*= s; z \*= s; }  // 加算の定義  Vector3d operator+(Vector3d v) { return Vector3d(x + v.x, y + v.y, z + v.z); }  // 減算の定義  ***Vector3d operator-(Vector3d v) { return Vector3d(x - v.x, y - v.y, z - v.z); }***  // 内積の定義  double operator\*(Vector3d v) { return x \* v.x + y \* v.y + z \* v.z; }  // 外積の定義  Vector3d operator%(Vector3d v) { return Vector3d(y \* v.z - z \* v.y, z \* v.x - x \* v.z, x \* v.y - y \* v.x); }  // 代入演算の定義  Vector3d& operator=(const Vector3d& v) { x = v.x; y = v.y; z = v.z; return (\*this); }  // 加算代入の定義  Vector3d& operator+=(const Vector3d& v) { x += v.x; y += v.y; z += v.z; return (\*this); }  // 減算代入の定義  ***Vector3d& operator-=(const Vector3d& v) { x -= v.x; y -= v.y; z -= v.z; return (\*this); }***  // 値を出力する  void print() { printf("Vector3d(%f %f %f)\n", x, y, z); }  };  // マイナスの符号の付いたベクトルを扱えるようにするための定義 例：b=(-a); のように記述できる  Vector3d operator-(const Vector3d& v) { return(Vector3d(-v.x, -v.y, -v.z)); }  // ベクトルと実数の積を扱えるようにするための定義 例： c=5\*a+2\*b; c=b\*3; のように記述できる  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)); }  // ベクトルを実数で割る操作を扱えるようにするための定義 例： c=a/2.3; のように記述できる  Vector3d operator/(const Vector3d& v, const double& k) { return(Vector3d(v.x / k, v.y / k, v.z / k)); }  // 球体の情報を格納するクラス  class Sphere {  public:  Vector3d position; // 中心位置  float color[3]; // 描画色  void setColor(float r, float g, float b) {  color[0] = r; color[1] = g; color[2] = b;  }  // この球体を描画するメンバ関数  void display() {  glPushMatrix(); // 現在のモデル変換行列を退避しておく  // 座標の平行移動とスケール変換を施して球体を描画する  glTranslated(position.x, position.y, position.z);  glScaled(2, 2, 2);  glutSolidSphere(1.0, 32, 32);  glPopMatrix(); // 退避していたモデル変換行列を戻す  }  };  // 3つの球体を準備しておく  Sphere g\_Sphere[3];  // 選択状態にある球体のID番号（0,1,2）を保持する。選択状態の球が無ければ-1とする。  int g\_SelectedSphereID = -1;  // クリックされた3次元座標を保持する  Vector3d g\_SelectedPos;  // ウィンドウサイズを保持する  int g\_WindowWidth = 512;  int g\_WindowHeight = 512;  // 選択した球体のID番号（0,1,2）を返す  // 選択した球体が無い場合は -1 を返す  int pickSphere(int x, int y) {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glEnable(GL\_DEPTH\_TEST);  // 照明効果を無くして単色で描画する  glDisable(GL\_LIGHTING);  // 3つの球体を描画する  for (int i = 0; i < 3; i++) {  // RGBのR成分に球体のIDを設定する(unsigned byte型)  glColor3ub(i, 0, 0);  g\_Sphere[i].display();  }  // ★授業スライドを参考に次のようなプログラムコードを追加する  // ★glReadPixels 関数をつかって、クリックした位置が何色であるか取得する  // ★取得した色を見て、どの球体を選択したか判定し、そのIDを return する。  ***GLubyte c[3]; //座標(x,y)の色を取得***  ***glReadPixels(x, y, 1, 1, GL\_RGB, GL\_UNSIGNED\_BYTE, c);***  ***return (c[0] == 255) ? -1 : (int)c[0]; // ★適切な値を返すようにする***  }  // 描画関数  void display() {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glEnable(GL\_DEPTH\_TEST);  glEnable(GL\_LIGHTING);  // 透視投影変換行列の設定  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(30.0, g\_WindowWidth / (float)g\_WindowHeight, 1.0, 100.0);  // カメラビュー座標への変換行列の設定  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  gluLookAt(0, 0, 30, 0, 0, 0, 0, 1, 0);  // 3つの球体を描画  for (int i = 0; i < 3; i++) {  // 球体ごとに色を変更する  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, g\_Sphere[i].color);  // 球体の描画を行う  g\_Sphere[i].display();  }  // 球が選択されている状態であれば、クリック座標に関する情報を表示する  if (g\_SelectedSphereID != -1) {  // 照明効果なしで単色描画  glDisable(GL\_LIGHTING);  glDisable(GL\_DEPTH\_TEST);  // クリック座標に点を描画  glColor3f(1, 0, 0);  glPointSize(5.f);  glBegin(GL\_POINTS);  glVertex3d(g\_SelectedPos.x, g\_SelectedPos.y, g\_SelectedPos.z);  glEnd();  // 文字を描画する位置の指定  glRasterPos3d(g\_SelectedPos.x, g\_SelectedPos.y, g\_SelectedPos.z);  // 表示する文字列の構築  // ※もし sprintf\_s でコンパイルエラーになる場合は sprintf を使うこと  char str[256];  sprintf\_s(str, "sphere[%d] (%lf, %lf, %lf)", g\_SelectedSphereID,  g\_SelectedPos.x, g\_SelectedPos.y, g\_SelectedPos.z);  // 文字列を1文字ずつ描画  for (int i = 0; str[i] != '\0'; i++) {  glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, str[i]);  }  }  glutSwapBuffers();  }  // ウィンドウのサイズが変更されたときの処理  void resize(int w, int h) {  if (h < 1) return;  glViewport(0, 0, w, h);  g\_WindowWidth = w;  g\_WindowHeight = h;  }  // マウスカーソル位置に基づく選択処理  void MousePick(int x, int \_y) {  printf("MousePick(%d, %d)\n", x, \_y);  // マウスクリックで得られる座標は左下原点なので OpenGLの座標系と合わせるためにy座標を反転する  const int y = g\_WindowHeight - \_y;  g\_SelectedSphereID = pickSphere(x, y);  // 球が選択されていないなら何もしない  if (g\_SelectedSphereID == -1) return;  // クリックした場所の座標値（3次元座標）を取得する  // ★授業スライドを参考に次のようなプログラムコードを追加する  // ★現在のモデルビュー行列を取得する  // ★現在の透視投影行列の取得を取得する  // ★現在のビューポートの情報を取得  // ★マウスクリックした位置の奥行き情報（z値）を取得する  // ★上記の情報に基づいて、クリックした位置のワールド座標を取得する  // ★取得した値は、g\_SelectedPos に格納しておく（←表示の時に使用する）  ***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; // (x, y) の奥行き値 (デプス) を取得***  ***glReadPixels(x, y, 1, 1, GL\_DEPTH\_COMPONENT, GL\_FLOAT, &z);***  ***double objX, objY, objZ; // ワールド座標を計算***  ***gluUnProject(x, y, z, M, P, V, &objX, &objY, &objZ);***  ***g\_SelectedPos.set(objX, objY, objZ);***    }  // マウスクリックのイベント処理  void mouse(int button, int state, int x, int y) {  if (state == GLUT\_DOWN) MousePick(x, y);  glutPostRedisplay();  }  // マウスドラッグのイベント処理  void motion(int x, int y) {  MousePick(x, y);  glutPostRedisplay();  }  // キーが押されたときのイベント処理  void keyboard(unsigned char key, int x, int y) {  switch (key) {  case 'q':  case 'Q':  case '\033':  exit(0); /\* '\033' は ESC の ASCII コード \*/  default:  break;  }  glutPostRedisplay();  }  void init() {  // 3つの球体の位置と色を設定しておく  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); // 背景の色を白に設定  // 照明の設定  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);  // 材質の設定  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;  } |

実行結果は添付した動画ファイルのようになった。