CG基礎 第五回レポート

201811395 山本雄太

課題(1)について、サンプルを実行しベクトルの操作について確認した。

課題(2)のソースコード

|  |
| --- |
| #define FREEGLUT\_STATIC  #define FREEGLUT\_STATIC  #include <cstdlib>  #include <cmath>  #include <vector>  #include <GL/glut.h>  // 2次元ベクトルを扱うためのクラス  class Vector2d {  public:  double x, y;  Vector2d() { x = y = 0; }  Vector2d(double \_x, double \_y) { x = \_x; y = \_y; }  void set(double \_x, double \_y) { x = \_x; y = \_y; }  // 長さを1に正規化する  void normalize() {  double len = length();  x /= len; y /= len;  }  // 長さを返す  double length() { return sqrt(x \* x + y \* y); }  // s倍する  void scale(const double s) { x \*= s; y \*= s; }  // 加算の定義  Vector2d operator+(Vector2d v) { return Vector2d(x + v.x, y + v.y); }  // 減算の定義  Vector2d operator-(Vector2d v) { return Vector2d(x - v.x, y - v.y); }  // 内積の定義  double operator\*(Vector2d v) { return x \* v.x + y \* v.y; }  // 代入演算の定義  Vector2d& operator=(const Vector2d& v) { x = v.x; y = v.y; return (\*this); }  // 加算代入の定義  Vector2d& operator+=(const Vector2d& v) { x += v.x; y += v.y; return (\*this); }  // 減算代入の定義  Vector2d& operator-=(const Vector2d& v) { x -= v.x; y -= v.y; return (\*this); }  // 値を出力する  void print() { printf("Vector2d(%f %f)\n", x, y); }  };  // 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  Vector2d operator-(const Vector2d& v) { return(Vector2d(-v.x, -v.y)); }  // 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  Vector2d operator\*(const double& k, const Vector2d& v) { return(Vector2d(k \* v.x, k \* v.y)); }  Vector2d operator\*(const Vector2d& v, const double& k) { return(Vector2d(v.x \* k, v.y \* 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  Vector2d operator/(const Vector2d& v, const double& k) { return(Vector2d(v.x / k, v.y / k)); }  // ================================================================================================  std::vector<Vector2d> g\_ControlPoints; // 制御点を格納する  // 表示部分をこの関数で記入  void display(void) {  glClearColor(1.0, 1.0, 1.0, 1.0); // 消去色指定  glClear(GL\_COLOR\_BUFFER\_BIT); // 画面消去  // 制御点の描画  glPointSize(5);  glColor3d(0.0, 0.0, 0.0);  glBegin(GL\_POINTS);  for (unsigned int i = 0; i < g\_ControlPoints.size(); i++) {  glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);  }  glEnd();  // 制御点を結ぶ線分の描画  glColor3d(1.0, 0.0, 0.0);  glLineWidth(1);  glBegin(GL\_LINE\_STRIP);  for (unsigned int i = 0; i < g\_ControlPoints.size(); i++) {  glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);  }  glEnd();  //ベジェ曲線の描画  glColor3d(0.0, 0.0, 1.0);  glLineWidth(2);  glBegin(GL\_LINE\_STRIP);  double x[100];  double y[100];  for (unsigned int i = 0; i < g\_ControlPoints.size(); i++) {  x[i] = g\_ControlPoints[i].x;  y[i] = g\_ControlPoints[i].y;  }  for (unsigned int s = 0; s < g\_ControlPoints.size() / 3; s++) {  for (double t = 0; t <= 1.0; t += 0.01) {  double xp, yp;  xp = (1 - t) \* (1 - t) \* (1 - t) \* x[s \* 3 + 0] + 3 \* t \* (1 - t) \* (1 - t) \* x[s \* 3 + 1] + 3 \* t \* t \* (1 - t) \* x[s \* 3 + 2] + t \* t \* t \* x[s \* 3 + 3];  yp = (1 - t) \* (1 - t) \* (1 - t) \* y[s \* 3 + 0] + 3 \* t \* (1 - t) \* (1 - t) \* y[s \* 3 + 1] + 3 \* t \* t \* (1 - t) \* y[s \* 3 + 2] + t \* t \* t \* y[s \* 3 + 3];  glVertex2d(xp, yp);  }  }  glEnd();  glutSwapBuffers();    }  void resizeWindow(int w, int h) {  h = (h == 0) ? 1 : h;  glViewport(0, 0, w, h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  // ウィンドウ内の座標系設定  // マウスクリックの座標と描画座標が一致するような正投影  glOrtho(0, w, h, 0, -10, 10);  glMatrixMode(GL\_MODELVIEW);  }  // キーボードイベント処理  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();  }  // マウスイベント処理  void mouse(int button, int state, int x, int y) {  if (state == GLUT\_DOWN) {  switch (button) {  case GLUT\_LEFT\_BUTTON:  // クリックした位置に制御点を追加  g\_ControlPoints.push\_back(Vector2d(x, y));  break;  case GLUT\_MIDDLE\_BUTTON:  break;  case GLUT\_RIGHT\_BUTTON:  // 末尾の制御点の削除  if (!g\_ControlPoints.empty()) {  g\_ControlPoints.pop\_back();  }  break;  default:  break;  }  glutPostRedisplay(); // 再描画  }  }  // メインプログラム  int main(int argc, char\* argv[]) {  glutInit(&argc, argv); // ライブラリの初期化  glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE); // 描画モードの指定  glutInitWindowSize(800, 800); // ウィンドウサイズを指定  glutCreateWindow(argv[0]); // ウィンドウを作成  glutDisplayFunc(display); // 表示関数を指定  glutReshapeFunc(resizeWindow); // ウィンドウサイズが変更されたときの関数を指定  glutKeyboardFunc(keyboard); // キーボードイベント処理関数を指定  glutMouseFunc(mouse); // マウスイベント処理関数を指定  glutMainLoop(); // イベント待ち  return 0;  } |

これにより書いた∞記号

