CG基礎 第九回レポート

201811395 山本雄太

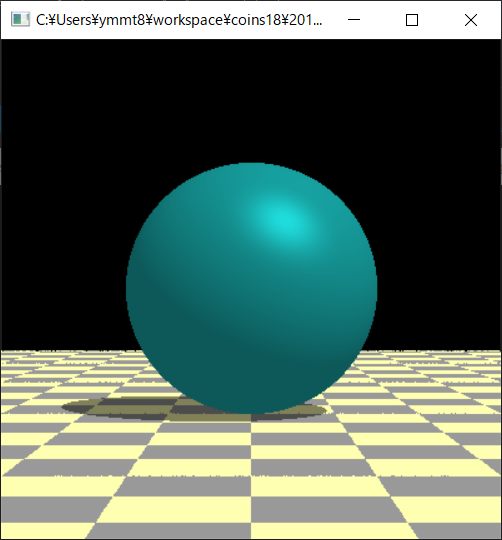
必須課題

実装したコードを以下に示す(サンプルコードから変更の無い部分は一部省略)

|  |
| --- |
| // 球体  class Sphere {  public:  Vector3d center; // 中心座標  double radius; // 半径  double cR, cG, cB; // Red, Green, Blue 値 0.0～1.0  Sphere(double x, double y, double z, double r,  double cr, double cg, double cb) {  center.x = x;  center.y = y;  center.z = z;  radius = r;  cR = cr;  cG = cg;  cB = cb;  }  // 点pを通り、v方向のRayとの交わりを判定する。  // 交点が p+tv として表せる場合の t の値を返す。交わらない場合は-1を返す  double getIntersec(Vector3d& p, Vector3d& v) {  // A\*t^2 + B\*t + C = 0 の形で表す  double A = v.x \* v.x + v.y \* v.y + v.z \* v.z;  double B = 2.0 \* (p.x \* v.x - v.x \* center.x +  p.y \* v.y - v.y \* center.y +  p.z \* v.z - v.z \* center.z);  double C = p.x \* p.x - 2 \* p.x \* center.x + center.x \* center.x +  p.y \* p.y - 2 \* p.y \* center.y + center.y \* center.y +  p.z \* p.z - 2 \* p.z \* center.z + center.z \* center.z -  radius \* radius;  double D = B \* B - 4 \* A \* C; // 判別式  if (D >= 0) { // 交わる  double t1 = (-B - sqrt(D)) / (2.0 \* A);  double t2 = (-B + sqrt(D)) / (2.0 \* A);  return t1 < t2 ? t1 : t2; // 小さいほうのtの値を返す  }  else { // 交わらない  return -1.0;  }  }  };  // 板。xz平面に平行な面とする  class Board {  public:  double y; // y座標値  Board(double \_y) {  y = \_y;  }  // 点pを通り、v方向のRayとの交わりを判定する。  // 交点が p+tv として表せる場合の t の値を返す。交わらない場合は負の値を返す  double getIntersec(Vector3d& p, Vector3d& v) {  if (fabs(v.y) < 1.0e-10) return -1; // 水平なRayは交わらない  double t = -1;  // ★ここで t の値を計算する  // ★ただしz座標が-3000より小さいなら交わらないものとする  t = -(150 + p.y) / v.y;  if (v.z \* t < -3000) {  t = -1;  }  return t;  }  // x と z の値から床の色を返す（格子模様になるように）  Vector3d getColorVec(double x, double z) {  // ★ x, z の値によって(1.0, 1.0, 0.7)または(0.6, 0.6, 0.6)のどちらかの色を返すようにする  int cx = -1, cz = -1;  if (x >= 0) {  cx = ((int)x / 100) % 2;  }  else {  cx = (((int)(-x) + 100) / 100) % 2;  }  if (z >= 0) {  cz = ((int)z / 100) % 2;  }  else {  cz = (((int)(-z) + 100) / 100) % 2;  }  if ((cx == 1 && cz == 1) || (cx == 0 && cz == 0)) {  return Vector3d(1.0, 1.0, 0.7);  }  else {  return Vector3d(0.6, 0.6, 0.6);  }  }  };  int halfWidth; // 描画領域の横幅/2  int halfHeight; // 描画領域の縦幅/2  double screen\_z = -1000; // 投影面のz座標  double Kd = 0.8; // 拡散反射定数  double Ks = 0.8; // 鏡面反射定数  double Iin = 0.5; // 入射光の強さ  double Ia = 0.5; // 環境光  Vector3d viewPosition(0, 0, 0); // 原点=視点  Vector3d lightDirection(-2, -4, -2); // 入射光の進行方向  // レンダリングする球体  Sphere sphere(0.0, 0.0, -1500, // 中心座標  150.0, // 半径  0.1, 0.7, 0.7); // RGB値  // 球体の置かれている床  Board board(-150); // y座標値を -150 にする。（球と接するようにする）  // 2つのベクトルの成す角のcos値を計算する  double getCos(Vector3d& v0, Vector3d& v1) {  return (v0.x \* v1.x + v0.y \* v1.y + v0.z \* v1.z) / (v0.length() \* v1.length());  }  // x, y で指定されたスクリーン座標での色(RGB)を colorVec のxyzの値に格納する  void getPixelColor(double x, double y, Vector3d& colorVec) {  // 原点からスクリーン上のピクセルへ飛ばすレイの方向  Vector3d ray(x - viewPosition.x, y - viewPosition.y, screen\_z - viewPosition.z);  ray.normalize(); // レイの長さの正規化  // レイを飛ばして球と交差するか求める  double t\_sphere = sphere.getIntersec(viewPosition, ray);  if (t\_sphere > 0) { // 球との交点がある  // ★前回の課題を参考に、球体の表面の色を計算で求め、colorVecに設定する  double Is = 0; // 鏡面反射光  double Id = 0; // 拡散反射光  Vector3d viewPositionToRayPoint = t\_sphere \* ray; // 視点からレイと球との交点へのベクトル  Vector3d sphereCenter(viewPosition.x, viewPosition.y, viewPosition.z - 1500); // 視点から球中心へのベクトル  Vector3d N = viewPositionToRayPoint - sphereCenter;  N.normalize(); // 法線ベクトル  lightDirection.normalize();  if (((-lightDirection) \* N) >= 0) {  Id = Iin \* Kd \* ((-lightDirection) \* N);  }  Vector3d V = t\_sphere \* ((Vector3d(0, 0, -1500)) - viewPosition);  V.normalize();  double a = (-lightDirection) \* N;  Vector3d R = lightDirection + (2 \* a \* N);  R.normalize();  double cosy = 0;  int npow = 10;  cosy = pow(R \* V, npow);  if ((-lightDirection) \* N >= 0) {  Is = Iin \* Ks \* cosy;  }  double I = Id + Is + Ia;  double r = std::min(I \* sphere.cR, 1.0); // 1.0 を超えないようにする  double g = std::min(I \* sphere.cG, 1.0); // 1.0 を超えないようにする  double b = std::min(I \* sphere.cB, 1.0); // 1.0 を超えないようにする  colorVec.set(r, g, b);  return;  }  // レイを飛ばして床と交差するか求める  double t\_board = board.getIntersec(viewPosition, ray);  if (t\_board > 0) { // 床との交点がある  // ★床の表面の色を設定する  // ★球の影になる場合は、RGBの値をそれぞれ0.5倍する  double tx = t\_board \* ray.x;  double ty = t\_board \* ray.y;  double tz = t\_board \* ray.z;  Vector3d point\_board(tx, ty, tz);  double r = board.getColorVec(tx, tz).x;  double g = board.getColorVec(tx, tz).y;  double b = board.getColorVec(tx, tz).z;  lightDirection.normalize();  Vector3d light = -lightDirection;  double u = sphere.getIntersec(point\_board, light);  if (u > 0) {  r = r \* 0.5;  g = g \* 0.5;  b = b \* 0.5;  }  colorVec.set(r, g, b);  return;  }  // 何とも交差しない  colorVec.set(0, 0, 0); // 背景色（黒）を設定する  }  // 描画を行う  void display(void) {  glClear(GL\_COLOR\_BUFFER\_BIT); // 描画内容のクリア  // ピクセル単位で描画色を決定するループ処理  for (int y = (-halfHeight); y <= halfHeight; y++) {  for (int x = (-halfWidth); x <= halfWidth; x++) {  Vector3d colorVec0;  Vector3d colorVec1;  Vector3d colorVec2;  Vector3d colorVec3;  Vector3d colorVec4;  Vector3d colorVec5;  Vector3d colorVec6;  Vector3d colorVec7;  Vector3d colorVec8;  // x, y 座標の色を取得する  getPixelColor((double)x , (double)y , colorVec0);  getPixelColor((double)x , (double)y + 0.334 , colorVec1);  getPixelColor((double)x , (double)y + 0.667 , colorVec2);  getPixelColor((double)x + 0.334 , (double)y , colorVec3);  getPixelColor((double)x + 0.334 , (double)y + 0.334 , colorVec4);  getPixelColor((double)x + 0.334 , (double)y + 0.667 , colorVec5);  getPixelColor((double)x + 0.667 , (double)y , colorVec6);  getPixelColor((double)x + 0.667 , (double)y + 0.334 , colorVec7);  getPixelColor((double)x + 0.667 , (double)y + 0.667 , colorVec8);  //取得した色で、描画色を設定する  double colorX = (colorVec0.x + colorVec1.x + colorVec2.x + colorVec3.x + colorVec4.x + colorVec5.x + colorVec6.x + colorVec7.x + colorVec8.x)/9;  double colorY = (colorVec0.y + colorVec1.y + colorVec2.y + colorVec3.y + colorVec4.y + colorVec5.y + colorVec6.y + colorVec7.y + colorVec8.y)/9;  double colorZ = (colorVec0.z + colorVec1.z + colorVec2.z + colorVec3.z + colorVec4.z + colorVec5.z + colorVec6.z + colorVec7.z + colorVec8.z)/9;  glColor3d(colorX, colorY, colorZ);  // (x, y) の画素を描画  glBegin(GL\_POINTS);  glVertex2i(x, y);  glEnd();  }  }  glFlush();  } |

実行結果

アンチエイリアシング実装前



アンチエイリアシング実装後

