CAMD LAB3

學號:609420042
姓名:陳邦寓

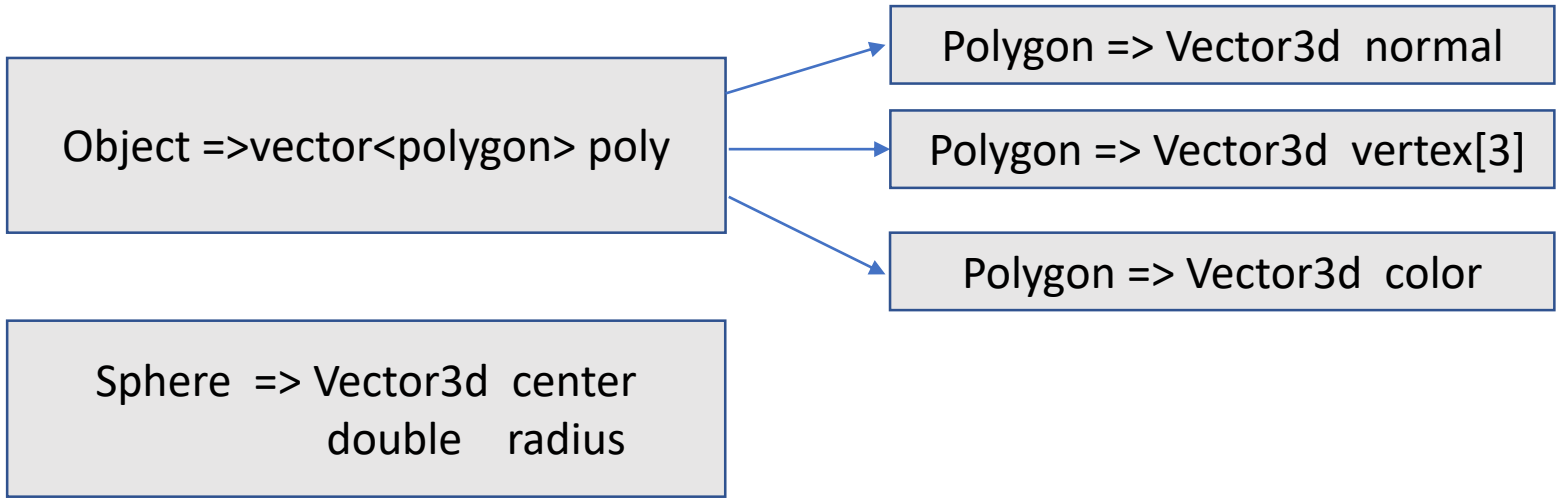
1. LAB objective

目標:1.用RayCasting繪出物件、計算出圓繪出

2.使用Lab1的矩陣轉換

3.讀入檔案放入->class裡面。

資料結構大概是這樣。



4.計算simplified_illumination_model值。

2. Describe your method

讀檔➔

讀檔和上次相同，只是這次只讀 Cube 和 QuadrangularCone

```
27  Object Modle(string name)
28  {
29      Object OBJ;
30      ifstream file(name,ios::in);
31      string temp;
32      polygon pol;
33
34      while (!file.eof())
35      {
36
37          file >> temp;
38          if (temp == "normal")
39          {
40              file >> pol.normal[0] >> pol.normal[1] >> pol.normal[2];
41          }
42          if (temp == "vertex" )
43          {
44              file >> pol.vertex[0][0] >> pol.vertex[0][1] >> pol.vertex[0][2];
45              pol.vertex[0][3] = 1;
46              file >> temp >> pol.vertex[1][0] >> pol.vertex[1][1] >> pol.vertex[1][2];
47              pol.vertex[1][3] = 1;
48              file >> temp >> pol.vertex[2][0] >> pol.vertex[2][1] >> pol.vertex[2][2];
49              pol.vertex[2][3] = 1;
50              OBJ.poly.push_back(pol);
51          }
52      }
53      return  OBJ;
54      file.close();
55
56  }
```

```
12  class polygon
13  {
14  public:
15      Vector3d normal;
16      Vector4d vertex[3];
17      Vector3d color;
18  };
19
20  class Object {
21  public:
22      vector<polygon> poly;
23  };
```

```
45  #pragma region Cube input
46  Object Cube_1 = Modle("Cube.txt");
47  #pragma endregion
48
49  #pragma region QuadrangularCone input
50  Object QuadrangularCone = Modle("Quadrangularcone.txt");
51  #pragma endregion
52
```

- 將檔案裡面的 normal 存到類的容器內，
- 將vector存到類的個別，[3]個容器內。

```cpp
Matrix4d EyetoWorld_matrix(Vector3d Eye_position,double X_coi,double Y_coi,double Z_coi)
{
    Matrix4d EyetoWorld_result;
    EyetoWorld_result.setIdentity();
    // ...
    P_coi.x() = X_coi;
    P_coi.y() = Y_coi;
    P_coi.z() = Z_coi;

    z_eye_direction_see_to_coi = -(P_coi - Eye_position);
    z_eye_direction_see_to_coi.normalize();//z軸正規化

    Vector3d Up_vector(0, 1, 0);

    Vector3d x_eye_direction_see_to_coi = Up_vector.cross(z_eye_direction_see_to_coi);//x軸向量。 y軸 cross z軸
    x_eye_direction_see_to_coi.normalize();//x軸正規化

    Vector3d y_eye_direction_see_to_coi = z_eye_direction_see_to_coi.cross(x_eye_direction_see_to_coi);//y軸向量

    Matrix3d RRR;
    RRR << x_eye_direction_see_to_coi,
        y_eye_direction_see_to_coi,
        z_eye_direction_see_to_coi;

    Vector3d W_E_displacement;
    Vector3d World_xyz = {0,0,0};
    W_E_displacement = Eye_position - World_xyz;

    EyetoWorld_result.block(0, 0, 3, 3) = RRR;
    EyetoWorld_result.block(0, 3, 3, 1) = W_E_displacement;
    return EyetoWorld_result;
}
```

$$\overrightarrow{P_w} = H_e^w \overrightarrow{P_e}$$

Review Lab1:

只有這裡要特別注意!和Lab1不一樣的地方。

$$H_e^w = \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} & \overrightarrow{O_w O_e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\hat{z} = \frac{\overrightarrow{O_e P_{C.O.I}}}{\left|\overrightarrow{O_e P_{C.O.I}}\right|}$$

$$\hat{x} = [0,1,0]^T \times \hat{z}$$

$$\hat{y} = \hat{z} \times \hat{x}$$

- 眼睛到世界矩陣，和Lab1的世界到眼睛矩陣大致相同。

轉換到

眼睛空間 → 世界空間

```cpp
Vector3d Rd_calculation(int x, int y, Cube &C1)
{
    Vector4d Ray = { (x - WIDTH / 2) * delta_w, (y - HEIGHT / 2) * delta_h, -f, 0 };
    Ray.normalize();
    //EyetoWorld_matrix(Vector3d Eye_position,double X_coi,double Y_coi,double Z_coi)
    Rd_Ray_transform_world = C1.EyetoWorld_matrix(Eye_position, 0, 0, 0) * Ray;
    Rd_unitvector = Rd_Ray_transform_world.head(3).normalized();
    return  Rd_unitvector;
}
```

計算Rd一個函式。函式參數x、y可以遍歷pixel ，可以傳到球 和 三角網格 做計算方向。

```cpp
double Ka = 0.4 ,Ia=0.8;
double Kd = 0.6, I;
double Ks = 0.9, Is;
Vector3d Lightsource = { -50.0, 50.0, 50.0 };
Vector3d Light, Mirror_Light, Eye_vector, H;
Vector3d id_normal;
double simplified_illumination_model(Vector3d normalvector, Vector3d Ray_result)
{

    id_normal = normalvector;
    id_normal.normalize();
    Light = Lightsource - Ray_result;//光源 減 每個有在圓內點
    Light.normalize();
    double result = id_normal.dot(Light);

    Mirror_Light = Lightsource - Ray_result;
    Eye_vector = Eye_position - Ray_result;
    H = Mirror_Light + Eye_vector;
    H.normalize();

    Is = pow(H.dot(normalvector.normalized()), 300);//次方越大，打到物體的反光圈越小

    I = std::min(1.0, Ka*Ia + Kd * std::max(0.0, result) + Ks * Is);//如果大於1，取最小值1，I介於0~1之間
    return I;
}
```

光罩模型，這次寫在一個函式。這樣三角網格 和 球 ，都可以套用此函式，和Lab2不一樣地方 ，加了鏡射效果。

```cpp
double ray_R[WIDTH][HEIGHT] = { 0 };//Red，初始化
double ray_G[WIDTH][HEIGHT] = { 0 };
double ray_B[WIDTH][HEIGHT] = { 0 };

void RayCasting_Background_Color(float Red, float Green, float Blue)
{
    for (int i = 0; i < WIDTH; i++)
    {
        for (int j = 0; j < HEIGHT; j++)
        {
            glColor3f(ray_R[i][j] = Red, ray_G[i][j] = Green, ray_B[i][j] = Blue);
            glVertex2d(i, j);
        }
    }
}
```

背景底色初始化，for迴圈遍歷所有pixel。

注意：要取最小值，回傳 I 是介於0~1之間

```cpp
void sphere(int x,int y, Vector3d Rd_unitvector, Vector3d Ray_result_sphere ,double &t_compare)
{
    double L_scalar;
    Vector3d L_vector, Eye_to_ballCenter, D, E;
    double D_scalar, a;
    Sphere s1;
    s1.center = { 0,0,0 };
    s1.radius = 5;

    Eye_to_ballCenter = s1.center - Eye_position;
    L_scalar = Eye_to_ballCenter.dot(Rd_unitvector);//純量
    L_vector = L_scalar * Rd_unitvector;//向量

    D = Eye_to_ballCenter - L_vector;//鄰邊 = 斜邊 - 底邊
    D_scalar = sqrt(pow(D.x(), 2) + pow(D.y(), 2) + pow(D.z(), 2));//鄰邊取大小

    if (D_scalar < s1.radius)//小於圓半徑，才放入buffer
    {
        a = sqrt(pow(s1.radius, 2) - (D_scalar, 2));//三角形底邊 純量

        double sphere_t = sqrt(pow(L_vector.x(), 2) + pow(L_vector.y(), 2) + pow(L_vector.z(), 2)) - a;//射線射到物體最短距離t (純量)

        if (t_compare > sphere_t)
        {
            Ray_result_sphere = Eye_position + sphere_t * Rd_unitvector;//碰到球射線
            sphere_normalvector = Ray_result_sphere - s1.center; //球法向量
            double photomask = simplified_illumination_model(sphere_normalvector, Ray_result_sphere);
            ray_R[x][y] = 0.0;
            ray_G[x][y] = photomask;
            ray_B[x][y] = 0.6;
        }
    }
}
```

```cpp
class Sphere
{
public:
    Vector3d center;
    double radius;
};
```

建立一個球的類，類內給圓心、半徑，可供給球的大小。

Sphere_t球的深度值只要減a，就可以判斷最近點。不需要加a去求背面那個點，因為看不到。

傳進來的if較大，就把小的t值，去計算，(最近的)實際碰到物體的射線

傳光罩模型進來

計算球函式。參數列表有傳t_compare，可比較深度， t_compare在main()的遍歷像素裡面，初始化很遠的地方我給1000，比較完後像Lab2一樣存入每個pixel。

計算三角網格射線

```
175  void RayCasting_mesh(int x, int y, Cube &C1, Object obj, Vector3d Rd_unitvector, double &t_compare)
176  {
177      Vector3d triangleMesh_normal, triangleMesh_vertex;
178      double d;
179      double t;
180      for (int a = 0; a < obj.poly.size(); a++)
181      {
182          double front_or_back_judge = Rd_unitvector.dot(obj.poly[a].normal);
183          if (front_or_back_judge > 0)//大於0在背面，則不做
184          {
185              continue;
186          }
187          int condition = 0;
188          triangleMesh_normal = obj.poly[a].normal;
189          triangleMesh_vertex = obj.poly[a].vertex[0].head(3);
190
191          d = triangleMesh_normal.dot(triangleMesh_vertex);
192          t = (d - triangleMesh_normal.dot(Eye_position)) / triangleMesh_normal.dot(Rd_unitvector);
193
194          Ray_result_sphere = Eye_position + t * Rd_unitvector;//向量
195
196          for (int b = 0; b < 3; b++)
197          {
198              Vector3d A1, A2;
199              Vector3d p0, p1, p2;
200              p0 = obj.poly[a].vertex[b].head(3);//當前點
201              p1 = obj.poly[a].vertex[(b + 1) % 3].head(3);//下一個點
202              p2 = obj.poly[a].vertex[(b + 2) % 3].head(3);//另外組合 下一個點
203
204              A1 = (p1 - p0).cross(Ray_result_sphere - p0);
205              A2 = (Ray_result_sphere - p0).cross(p2 - p0);
206              double A3;
207              A3 = A1.dot(A2);
208              if (A3 > 0)//判斷是否在三角形內
209              {
210                  condition++;//b要跑3次，都成立才能存到 buffer
211              }
212          }
213          if (condition == 3 )
214          {
215              if (t_compare > t)//比較深度
216              {
217                  t_compare = t;//更新到=>最小值
218                  Ray_result_sphere = Eye_position + t_compare * Rd_unitvector;//向量
219                  double photomask = simplified_illumination_model(triangleMesh_normal, Ray_result_sphere);
220                  ray_R[x][y] = photomask * obj.poly[a].color.x();
221                  ray_G[x][y] = photomask * obj.poly[a].color.y();
222                  ray_B[x][y] = photomask * obj.poly[a].color.z();
223              }
224          }
225      }
226  }
```

先判斷在物體背面不做，就是dot()之後值，
如果介於0 ~ -1就是背面，就不做

注意:這裡也要計算Ray_result_sphere值

上面condition有加到3才做。
有加到3表示射線有射到平面 也有 射到三角網格內。

上面成立才去比較深度值，
再計算出Ray_result_sphere值，
最後再把它存到buffer裡面

main()

```
391     RayCasting_Background_Color(1.0, 1.0, 1.0);//R G B
392
393     for (int i = 0; i < Cube_1.poly.size(); i++)//遍歷，初始化顏色poly
394     {
395         Cube_1.poly[i].color.x() = 1.0;
396         Cube_1.poly[i].color.y() = 0.0;
397         Cube_1.poly[i].color.z() = 0.0;
398     }
399
400     for (int i = 0; i < QuadrangularCone.poly.size(); i++)//遍歷，初始化顏色poly
401     {
402         QuadrangularCone.poly[i].color.x() = 0.0;
403         QuadrangularCone.poly[i].color.y() = 0.0;
404         QuadrangularCone.poly[i].color.z() = 1.0;
405     }
406
407
408     for (int x = 0; x < WIDTH; x++)//把所有Z-buffer存好的值繪出，遍歷所有pixel
409     {
410         for (int y = 0; y < HEIGHT; y++)
411         {
412             double t_compare = 1000;//深度比較初始化
413
414             Rd_calculation(x, y, C1);
415             //sphere
416             sphere(x, y, Rd_unitvector, Ray_result_sphere, t_compare);
417             //mesh
418             RayCasting_mesh(x, y, C1, Cube_1, Rd_unitvector, t_compare);
419             RayCasting_mesh(x, y, C1, QuadrangularCone, Rd_unitvector, t_compare);
420
421         }
422     }
```

將背景底色函式，放入main()

將每個物件初始化顏色，main()

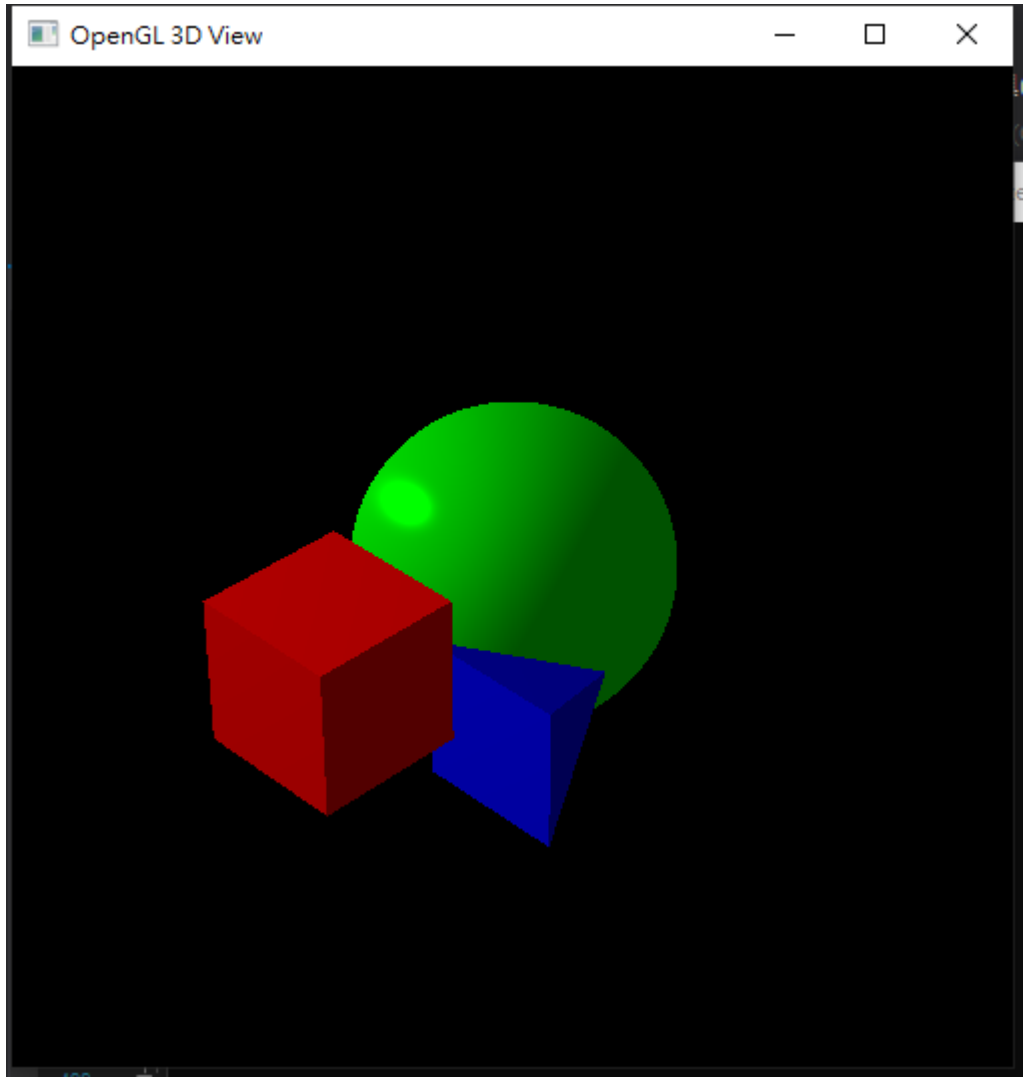
將Rd函式，放入main()

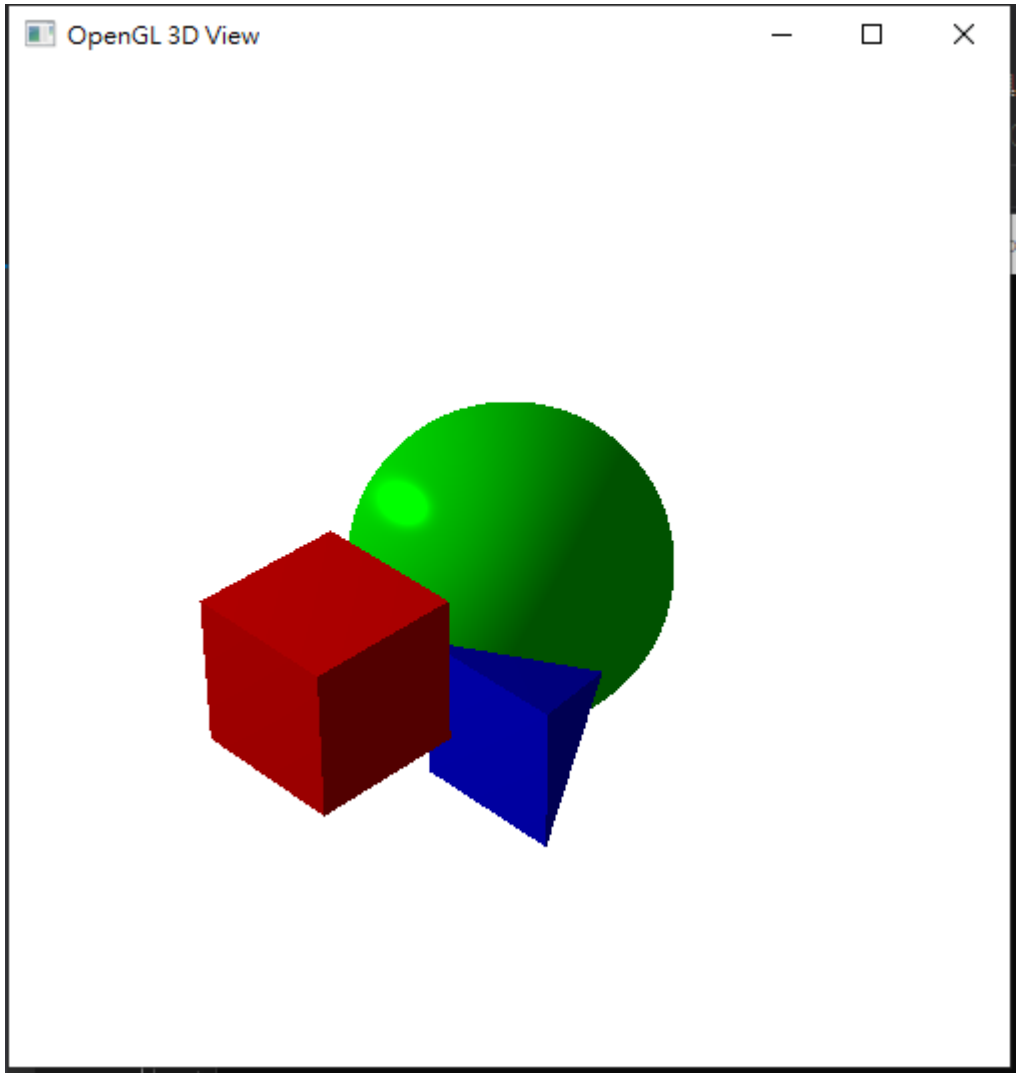
將sphere函式、RayCasting_mesh函示，放入main()

```
634 void display()
635 {
636     glClear(GL_COLOR_BUFFER_BIT);
637     //glColor3f(0.6, 1.0, 0);//紅線畫
638     glBegin(GL_POINTS);
639
640     for (int x = 0; x < WIDTH; x++)//把所有Z-buffer存好的值繪出，遍歷所有pixel
641     {
642         for (int y = 0; y < HEIGHT; y++)
643         {
644             glColor3f(ray_R[x][y], ray_G[x][y], ray_B[x][y]);
645             glVertex2d(x, y);
646         }
647     }
```

最後將所有存在buffer裡面的顏色、深度值，在display函示裡面繪出。

3. show(display) result

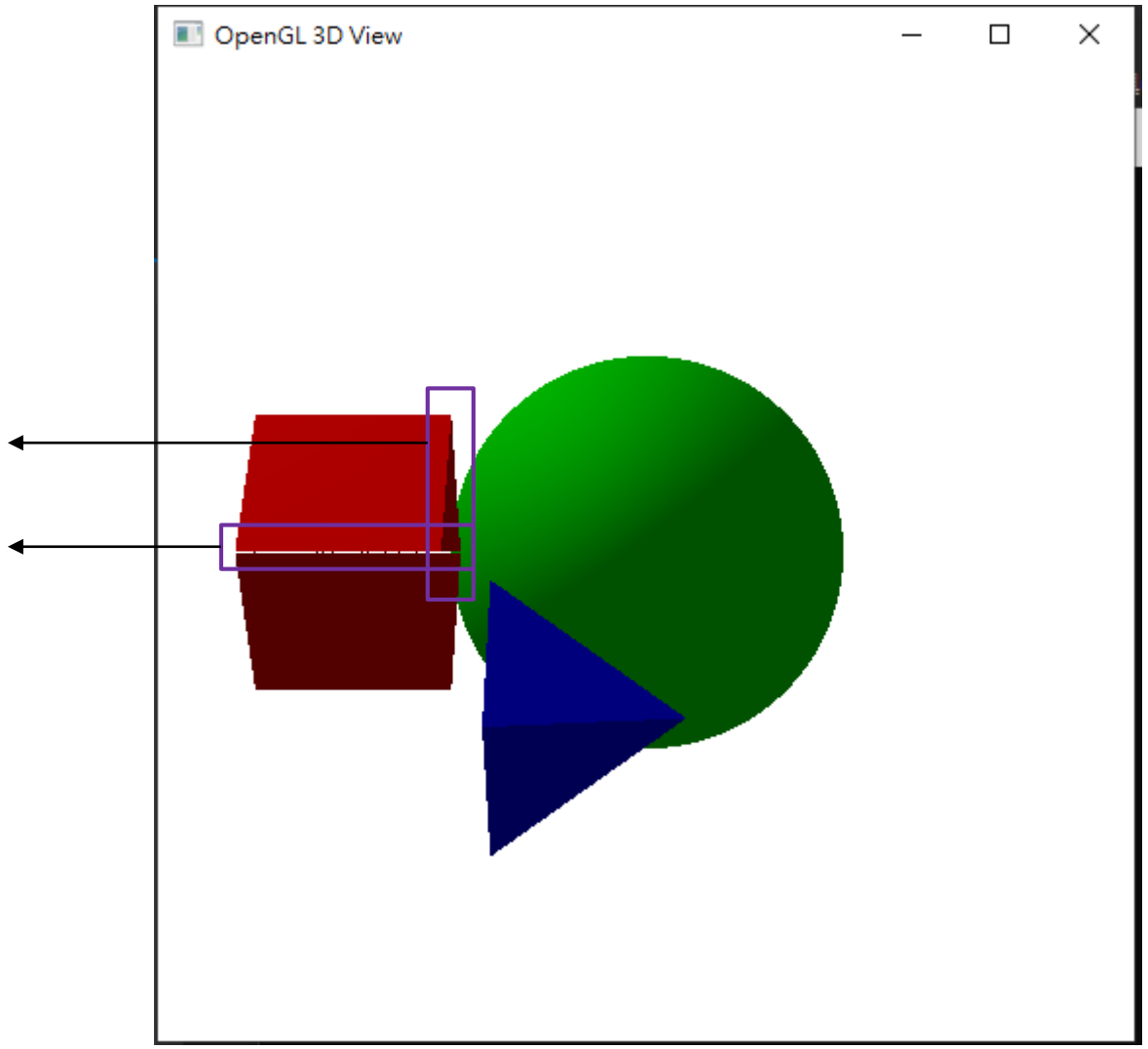




4. Discussion

1. 我寫程式能力，仍然需要加油。

2.這次作業在Z軸等於0{座標(50,50,0)}
，繪出時還有點問題，待改進。



3.最後感謝學長的ppt和同學的教導。

感謝觀看！