Computer Graphics, Task 5 - Guidelines

Paweł Aszklar
P.Aszklar@mini.pw.edu.pl

Warsaw, May 31, 2016

1 Introduction

This document contains guidelines for Task 5 which consists of displaying a representation of a three dimensional scene in the program window. For drawing, only library functions for coloring individual pixels and drawing line segments can be used. All other functionality, described later in this document, including scene modeling, projecting those models onto the program window, and calculating colors of the resulting pixels needs to be implemented by yourself. In particular, you should not use such functionality build into 3D rendering libraries such as Direct3D and OpenGL.

2 Scene Modeling

Scene presented by the program will consist of several types of abstract objects. They need to be modeled in a way that will simplify creation of the final image. The scene may include following types of objects:

- **Shapes** — representing shapes displayed on the screen.
- **Light sources** — if present, they provide lighting for the scene. They are not drawn themselves, but may affect the colors of pixels on the surface of an object drawn on the screen.
- **Camera** — defines the position and orientation of the point of view from which the scene is being observed.

Creation of models for the solids, that are present in the scene, will often require the use of points and vectors describing positions and directions in a 3D space. Those points and vectors will often need to be expressed in and converted between different coordinate systems. It is then vitally important to always remember which coordinate system we are currently working in. For the purpose of this task, we will assume that all coordinate systems used are right-handed (e.g. for the coordinate system of the camera, the X axis points to the right, the Y axis points upwards, and the Z axis points towards us, i.e. away from the screen — for reference in a left-handed coordinate system the Z axis would point into the screen instead).

2.1 Shapes

In this task the only shapes present in the scene will be spheres, each define by position of its center (expressed in the coordinate system of the scene), its radius, and its material coefficients (as described in section 5.2).

Bearing in mind the representation of transformations described in section 3, when creating the mesh, it is better to already store points **p** with four coordinates, as follows:

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

2.2 Light sources

There are three types of light sources that can be placed in the scene. Each light source will have its color (usually white). Other parameters depend on the light source type:

- **Point light** defined by its position - a point in the global coordinate system
- **Directional light** defined by its direction - a vector in the global coordinate system
- **Spot light** defined by position, direction (expressed in global coordinate system) and a coefficient describing the size of the cone of light.

Description of those parameters and how they affect the lighting in the scene can be found in section 5.1.

2.3 Camera

Camera describes the point-of-view (position and orientation) used to observe the scene (i.e. how the image plane of the window is positioned in relation to the rest of the scene). To properly display the scene on the screen we need to transform all objects from the coordinate system of the scene (i.e. the global coordinate system) to the coordinate system of the camera (see section 3.2)

3 Transformations

To represent transformations mentioned in the previous section in an uniform manner, each three-dimensional point $\mathbf{p}$ an vector $\mathbf{v}$ needs to be expressed using so called *affine*, aka. *homogeneous*, coordinates. Using this representation, each point and vector has four elements, first three are equal to its coordinates in 3D, and the fourth one is equal to 1 for points and 0 for vectors.

$$\mathbf{p} = [p_x, p_y, p_z, 1]^T$$
$$\mathbf{v} = [v_x, v_y, v_z, 0]^T$$

This will allow us to express all transformations as 4×4 matrices. Transformation of a point or vector will be performed by multiplying such matrix by a 4-element vector:

$$v' = Mv$$

3.1 Transformation between coordinate systems

Transformation between two coordinate systems is expressed by a so called *affine* transformation matrix in the form of:

$$M = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

First three columns define the change of orientation of the axes. The last column describes the shift of point of origin. This explains the choice of 1 as the fourth coordinate for points and 0 for vectors, since coordinates of vectors are only affected by the change in orientation of axes, the coordinates of points however, are also modified by the translation of the origin, described by the last column.

3.1.1 Geometric interpretation of affine transformation

Elements of affine transformation matrix can be interpreted as follows:

$$M = \begin{bmatrix} X_x^{src} & Y_x^{src} & Z_x^{src} & 0_x^{src} \\ X_y^{src} & Y_y^{src} & Z_y^{src} & 0_y^{src} \\ X_z^{src} & Y_z^{src} & Z_z^{src} & 0_z^{src} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\mathbf{X}^{src}$, $\mathbf{Y}^{src}$ i $\mathbf{Z}^{src}$ define coordinates of unit vectors of the axes of the source coordinate system expressed in destination coordinate system. Similarly $\mathbf{0}^{src}$ describes coordinates of point of origin of the source coordinate system expressed in the destination one.

3.1.2 Construction of affine transformation matrix

Construction of transformation matrix, where the axes and the origin of the source coordinate system are given in the destination one is trivial. Often however, we encounter the opposite case where axes $\mathbf{X}^{dst}$, $\mathbf{Y}^{dst}$, $\mathbf{Z}^{dst}$ and the origin $\mathbf{0}^{dst}$ of the destination coordinate system are expressed using source coordinates. Those can be used to construct the reverse transformation M^{-1}:

$$M^{-1} = \begin{bmatrix} X_x^{dst} & Y_x^{dst} & Z_x^{dst} & 0_x^{dst} \\ X_y^{dst} & Y_y^{dst} & Z_y^{dst} & 0_y^{dst} \\ X_z^{dst} & Y_z^{dst} & Z_z^{dst} & 0_z^{dst} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To calculate the desired transformation M, we need to invert the above matrix.

That matrix can be decomposed to:

$$M_{-1} = TO$$

where:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0_x^{dst} \\ 0 & 1 & 0 & 0_y^{dst} \\ 0 & 0 & 1 & 0_z^{dst} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$O = \begin{bmatrix} X_x^{dst} & Y_x^{dst} & Z_x^{dst} & 0 \\ X_y^{dst} & Y_y^{dst} & Z_y^{dst} & 0 \\ X_z^{dst} & Y_z^{dst} & Z_z^{dst} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If both coordinate systems are orthonormal (i.e. axes vectors in each are unit length and perpendicular), then:

$$T^{-1} = -T$$

$$O^{-1} = O^T$$

and as a result:

$$M = \left(M^{-1}\right)^{-1} = (TO)^{-1} = O^{-1}T^{-1} = O^T\,(-T)$$

Multiplying the two matrices gives us:

$$M = \begin{bmatrix} X_x^{dst} & X_y^{dst} & X_z^{dst} & \mathbf{X}^{dst} \cdot \mathbf{0}^{dst} \\ Y_x^{dst} & Y_y^{dst} & Y_z^{dst} & \mathbf{Y}^{dst} \cdot \mathbf{0}^{dst} \\ Z_x^{dst} & Z_y^{dst} & Z_z^{dst} & \mathbf{Z}^{dst} \cdot \mathbf{0}^{dst} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\mathbf{A} \cdot \mathbf{B}$ is a three-dimensional dot product of two vectors.

3.1.3 Combinations of basic transformation

Positioning an object (whose position and orientation is defined by a matrix of transformation from the local coordinate system of the model to the global coordinate system of the scene) using the above method may not be the most convenient. Often a more intuitive approach would be to position the object using a series of basic transformations, such as rotations around axes, and translations (shifting of position). We can easily define affine transformation matrices for those basic operations. Combination of those operations will be equivalent to multiplication of their respective matrices.

Rotation matrices We will define three variants of rotation matrix, one for rotation around each of the axes. For each the parameter α will define the angle of rotation.

$$R_X(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_Y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_Z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation matrix Translation, usually represented by a translation vector $\mathbf{t} = [t_x, t_y, t_z]^T$, can also be expressed as an affine matrix:

$$T(\mathbf{t}) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Combination of transformations Combination of a series of consecutive transformations (in order first to last) $M_1, \ldots, M_n$ can be expressed by a single matrix M obtained by multiplying matrices of those transformations as follows:

$$M = M_n \cdots M_1$$

In that case the first transformation applied will be M_1, the second will be M_2, etc. *Warning!* Combination of transformation, similarly to matrix multiplication is not commutative, and thus the order of operations is important.

3.2 Construction of camera (view) matrix

Camera matrix (aka. view matrix) can be created in many different ways, depending on the parameters used to control it. In this example we will show a formula for camera matrix defined by:

- **cPos** — position of the camera expressed as a point in the coordinate system of the scene
- **cTarget** — position of camera target (i.e. a point the camera is "looking at") expressed in coordinate system of the scene
- **cUp** — auxiliary vector pointing "up" also expressed in coordinate system of the scene, which defines camera orientation (*Warning!* It should not be parallel to a line going through points **cPos** and **cTarget**)

Based on those parameters we can define an orthonormal local coordinate system of the camera (expressed in the coordinate system of the scene):

$$\mathbf{cZ} = \frac{\mathbf{cPos} - \mathbf{cTarget}}{\|\mathbf{cPos} - \mathbf{cTarget}\|}$$

$$\mathbf{cX} = \frac{\mathbf{cUp} \times \mathbf{cZ}}{\|\mathbf{cUp} \times \mathbf{cZ}\|}$$

$$\mathbf{cY} = \frac{\mathbf{cZ} \times \mathbf{cX}}{\|\mathbf{cZ} \times \mathbf{cX}\|}$$

where $\mathbf{A} \times \mathbf{B}$ is a three-dimensional cross product of two vectors and $\|\mathbf{A}\|$ is vectors length.

cPos is the origin of the camera coordinate system, and its axes are defined by unit length perpendicular vectors **cX**, **cY**, **cZ**. Using the method described in 3.1.2 we can calculate the matrix as follows:

$$M = \begin{bmatrix} cX_x & cX_y & cX_z & \mathbf{cX} \cdot \mathbf{cPos} \\ cY_x & cY_y & cY_z & \mathbf{cY} \cdot \mathbf{cPos} \\ cZ_x & cZ_y & cZ_z & \mathbf{cZ} \cdot \mathbf{cPos} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4 Ray-casting algorithm

This section will present an algorithm for drawing 3D shapes on a screen using ray-casting. This method involves casting rays from the camera position through each pixel of the image. From all intersections of that ray with shapes in the scene we select the closest one. Based on the position of that intersection and the properties of the shape the ray intersected, we can calculate the color, that should be stored in the pixel the ray was cast through.

4.1 Ray construction

Ray is a half-line starting at some point $\mathbf{p}$ going in a direction defined by some vector $\mathbf{v}$. Rays going through the pixels of the image can be most easily expressed in the coordinate system of the camera. Starting point for each ray in this coordinate system will be:

$$\mathbf{p}' = [0, 0, 0, 1]^T$$

Rays going through pixels of the screen correspond to rays in the camera coordinate system that go through a virtual rectangular window perpendicular to the Z axis. The ratio between width w and height h of that window is equal to a (this ratio should be equal to the aspect ratio of the bitmap on which we are drawing), and the distance of this window from the camera should provide a horizontal field of view of θ:

$$a = \frac{w}{h} = \frac{screen_{width}}{screen_{height}}$$

where w is the length of the window edge parallel to X axis and h is the length of the window edge parallel to Y axis. Additionally Z intersects the center of the window.

There are infinite number of such windows that satisfy listed requirements. To make calculations easier we will chose one where $w = 2$. Construction of such window is illustrated in figure 1.

Distance between the window and camera is equal to:

$$d = \mathrm{tg}\frac{\theta}{2}$$

For each pixel we need to determine the corresponding point in the window:

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ -d \\ 1 \end{bmatrix}, \; x \in [-1, 1]\,, \; y \in [-a, a]$$

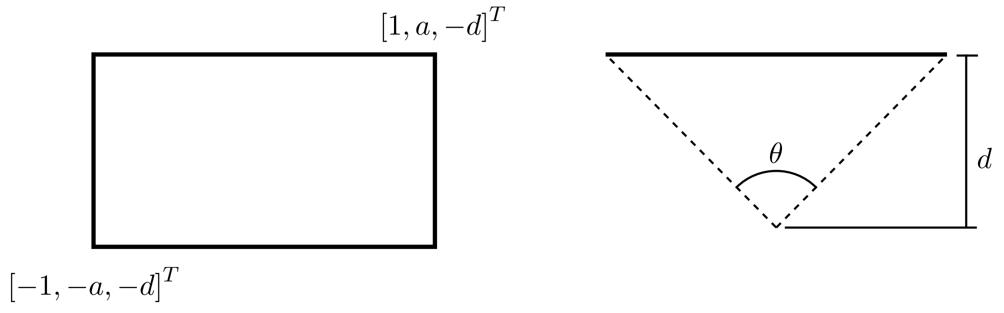

Figure 1: Ray construction

The exact mapping depends on the bitmap, but it should be done so that bottom-left pixel maps to $x = -1, y = -1$ and the top-right pixel maps to $x = 1, y = 1$.

From that it is easy to calculate ray direction $\mathbf{v}'$ as follows:

$$\mathbf{v}' = \frac{\mathbf{q} - \mathbf{p}'}{\|\mathbf{q} - \mathbf{p}'\|}$$

Since the positions of the shapes and lights are defined in the coordinate system of the scene, we need to transform the ray to it. Let M be the camera matrix. Ray in scene coordinate system is given by:

$$\mathbf{p} = \mathbf{M}_{-1}\mathbf{p}'$$

$$\mathbf{v} = \mathbf{M}_{-1}\mathbf{v}'$$

Inverted camera matrix can be easily calculated based on information presented in sections 3.1.1, 3.1.2 and 3.2.

4.2 Intersections

All points along ray $(\mathbf{p}, \mathbf{v})$ are given by the formula:

$$\mathbf{p}_t = \mathbf{p} + t\mathbf{v},\ t \in [0, +\infty)$$

Since all potential intersections will lie on the ray, for each intersection we need only to know the value of parameter t to calculate its position.

To find the closest intersection of the ray with scene elements, we need to find one with the lowest positive value of t. Besides the value of t for that intersection it is important to remember the position of the intersection $\mathbf{p}_t$, vector normal to the surface of the object at the intersection point $\mathbf{n}_t$ and material coefficient of the intersected surface, as they will be required to calculate the color of the pixel (see section 5).

4.2.1 Intersections with a sphere

Let the sphere be defined by

- $\mathbf{p}_s$ - center
- r - radius.

Intersection of a ray with sphere surface must satisfy the following equation:

$$\|\mathbf{p}_t - \mathbf{p}_s\| = \|\mathbf{p} + t\mathbf{v} - \mathbf{p}_s\| = r$$

Squaring both sides of the equation produces:

$$(\mathbf{p} + t\mathbf{v} - \mathbf{p}_s) \cdot (\mathbf{p} + t\mathbf{v} - \mathbf{p}_s) = r^2$$

Equation can be expanded into:

$$t^2 + 2t\mathbf{v} \cdot (\mathbf{p} - \mathbf{p}_p) + \|\mathbf{p} - \mathbf{p}_p\|^2 - r^2 = 0$$

If this quadratic equation has two solutions, we select the smaller value of the two. If equation has no solutions, or solution (or one of the solutions) is negative, the ray doesn't intersect the sphere.

If the intersection exists and its position is $\mathbf{p}_t$, the normal vector to the sphere at that point is equal to:

$$\mathbf{n}_t = \frac{\mathbf{p}_t - \mathbf{p}_s}{\|\mathbf{p}_t - \mathbf{p}_s\|}$$

4.3 Coloring pixels

From all found intersections of a ray with scene elements, we only need the closest one, i.e. the one with smallest positive value of t. Based on the position of the intersection, vector normal to the surface at intersection point and surface material coefficients we can use Phong illumination model (as described in section 5) to calculate the color of the pixel. If no intersection was found, the corresponding pixel should be filled with a background color.

5 Phonga illumination model

What follows is an approximated model of interaction of surfaces with light, known as Phong illumination model. It allows us to find the intensity (color) of the light reflected by the surface at the given point in a given direction to the observer.

The intensity is described by a three-component vector $[r, g, b]^T$, representing red, green, and blue component of the reflected light. For the purpose of following calculations, we assume that each component is a real value in range of $[0, 1]$, and vector $[1, 1, 1]$ represents white color.

Firstly, we will introduce parameters and coefficients defining the light sources and reflective properties of materials the surfaces are made of.

5.1 Light sources

Phong illumination model describes three types of light sources: point, directional, and spot lights. Each light is described by its color. In addition to that point light is defined by its position and illuminates objects equally in all directions. Directional light is not attached to any given point, but instead is defined by a direction, meaning, that for any point in the scene, the light comes from the same direction. Spot lights are described by both position and direction. Points illuminated the most lie on a halfline starting from its position and going in its direction. The further the point is from that line, the less illuminated it is by this light source.

For the i^{th} light source, let:

- $\mathbf{p}_i$ be light source position expressed in global coordinate system (available only for point and spot lights)
- $\mathbf{d}_i$ be light source direction vector, expressed in global coordinate system (available only for directional and spot lights)
- I_i^p be light source base intensity (color — available for all light sources)
- r_i be spotlight focus coefficient (available only for spot lights; recommended values $\sim 10 - 100$).

5.2 Materials

In the Phong illumination model, the light reflected from a surface has three components. The first one is ambient reflection which is constant, emits equally in all directions and is independent of the relative positions of the surface, light sources and the observer. It represents light scattered uniformly throughout the whole scene. The second component is diffuse reflection. It consist of light coming from the light source that is reflected by the surface equally in all directions. The amount of light reflected depends on the angle between the normal vector and the vector pointing to the light source. It imitates matte surfaces. Third component, called specular reflection, where the intensity of the emitted light is the strongest in the direction of the oncoming light reflected of the surface. It imitates shiny surfaces.

For each different type of reflection, material of the surface should contain a three-element vector of coefficients in range of $[0, 1]$, representing the amount of red, green, and blue light reflected by the surface. In addition, for the specular reflection another scalar coefficient is needed, that describes the focus of this reflection. For a given surface let:

- $\mathbf{k}_a = \left[k_a^r, k_a^g, k_a^b\right]^T$ be a vector of ambient reflection coefficients
- $\mathbf{k}_d = \left[k_d^r, k_d^g, k_d^b\right]^T$ be a vector of diffuse reflection coefficients

- $\mathbf{k}_s = \left[k_s^r, k_s^g, k_s^b\right]^T$ be a vector of specular reflection coefficients
- m be specular focus coefficient

5.3 Calculating point color

Let us assume that we have a point $\mathbf{p}$ in global coordinates that lies on a surface with material coefficients as described in section 5.2 and that the vector normal to the surface at $\mathbf{p}$ is equal to $\mathbf{n}$. To calculate the point color as observed by the camera we need to know:

- $\mathbf{I}_a$ - intensity (color) of ambient light
- $\mathbf{p}_c$ - position of the camera in global coordinates

Formula for the color involves calculating the sum of ambient reflection, and, for each light source, the diffuse and specular reflections, as follows:

$$\mathbf{I} = \mathbf{I}_a\mathbf{k}_a + \sum_i \left(\mathbf{k}_d\mathbf{I}_i \max\left(\langle\mathbf{n}, \mathbf{l}_i\rangle, 0\right) + \mathbf{k}_s\mathbf{I}_i \max\left(\langle\mathbf{v}, \mathbf{r}_i\rangle, 0\right)^m\right)$$

where vector $\mathbf{v}$ defines a direction from the point on the surface to the camera:

$$\mathbf{v} = \frac{\mathbf{p}_c - \mathbf{p}}{\|\mathbf{p}_c - \mathbf{p}\|}$$

vector $\mathbf{l}_i$ defines direction from the point on the surface to the i^{th} light source:

- for point and spot lights

$$\mathbf{l}_i = \frac{\mathbf{p}_i - \mathbf{p}}{\|\mathbf{p}_i - \mathbf{p}\|}$$

- for directional lights

$$\mathbf{l}_i = -\mathbf{d}_i$$

vector $\mathbf{r}_i$ is a direction the light is coming from reflected by the surface at the given point:

$$\mathbf{r}_i = 2\langle\mathbf{n}, \mathbf{l}_i\rangle\mathbf{n} - \mathbf{l}_i$$

and $\mathbf{I}_i$ is the intensity of the light coming to the point from the i^{th} light source:

- for point and directional lights

$$\mathbf{I}_i = \mathbf{I}_i^p$$

- for spot lights

$$\mathbf{I}_i = \mathbf{I}_i^p \max\left(\langle-\mathbf{d}_i, \mathbf{l}_i\rangle, 0\right)$$