Introduction

What are the graphics systems?

A system consisting of an input device (mouse, joystick, …), processor (used to be CPUs, today GPUs), frame buffer and output device (CRTs and flat-screen technologies-LED, LCD, Plasma Panel).

What is the framebuffer?

Pixels of the image are stored in a memory area called a framebuffer, a portion of GPU memory.

What is resolution?

Number of pixels in the framebuffer.

What is depth?

Number of bits per pixel.

1-bit-deep means black and white.
8-bit-deep means 256 colours (gray scale / monochrome)
24-bit-deep means RGB-colour system: we have red, green, blue channels and for each channel there are 256 shades/colours.

What is rasterization (scan conversion)?

Taking graphic definitions (line, circle, …) and determining the pixel values to be displayed in the framebuffer.

We say that graphics systems are raster based? What does 'raster' mean?

Raster is an array of pixels (picture elements). Pixels (so raster) are stored in the framebuffer.

What are the elements of image formation? Or what do we need?

Objects (what does a viewer see), viewer (who does see the objects) and light source/sources (by the help of what a viewers see the objects).

How are the object geometries defined?

By sets of vertices. Most APIs provide sets of primitives (points, line segments, triangles, …). We obtain objects using primitives. Scenes are made of objects.

WebGL vs. OpenGL

WebGL: web-based, pragrammed in Java script, has fewer features, programmable pipeline (no fixed function pipeline)
OpenGL: desktop applications, written in C, has many features, includes fixed function pipeline

What is graphics pipeline?

a set of data processing elements connected in series, output of one element is the input of the next one.

to begin calculations for a new vertex, the previous vertex's calculations are not waited to finish.

Vertex processor: does coordinate transformations and color computations on each vertex.

Clipper and primitive assembler: clipping determines parts of objects that will be in the view. Clipping must be done at a primitive level, not vertex level, so we must also assemble vertices into primitives.

Rasterizer: converts vertices (output of primitive assembler) to fragments (potential pixels, operations can still happen on them). Fragments contain colour, depth, point size information.

Fragment processor: takes fragments as input and updates the pixels in the framebuffer. No operations can be done on pixels (it is the difference between a fragment and a pixel). Pixels contain only colour information.

Fixed function vs. programmable pipelines

Fixed function: graphics implemented through a set of pre-defined functions, programmer is limited.

Programmable: graphics hardware is programmable through shader programs implemented in a shader language (like GLSL), provide new possibilities for graphics.

Transformations

What are the geometric transformations?

Translation: moves the object. No deformation. Shape, size, and orientation remain the same.

Rotation: moves the object. No deformation. Shape and size remain the same.

Scaling: alters the size of an object. Any positive value can be used as scaling factor. Values less than 1 reduce the size of the object. Values greater than 1 enlarge the object. If scaling factor is 1 then the object stays unchanged.

Reflection: produces a mirror image of an object. Rotating the object 180 degrees around the reflection axes.

Shearing: distorts the shape of an object.

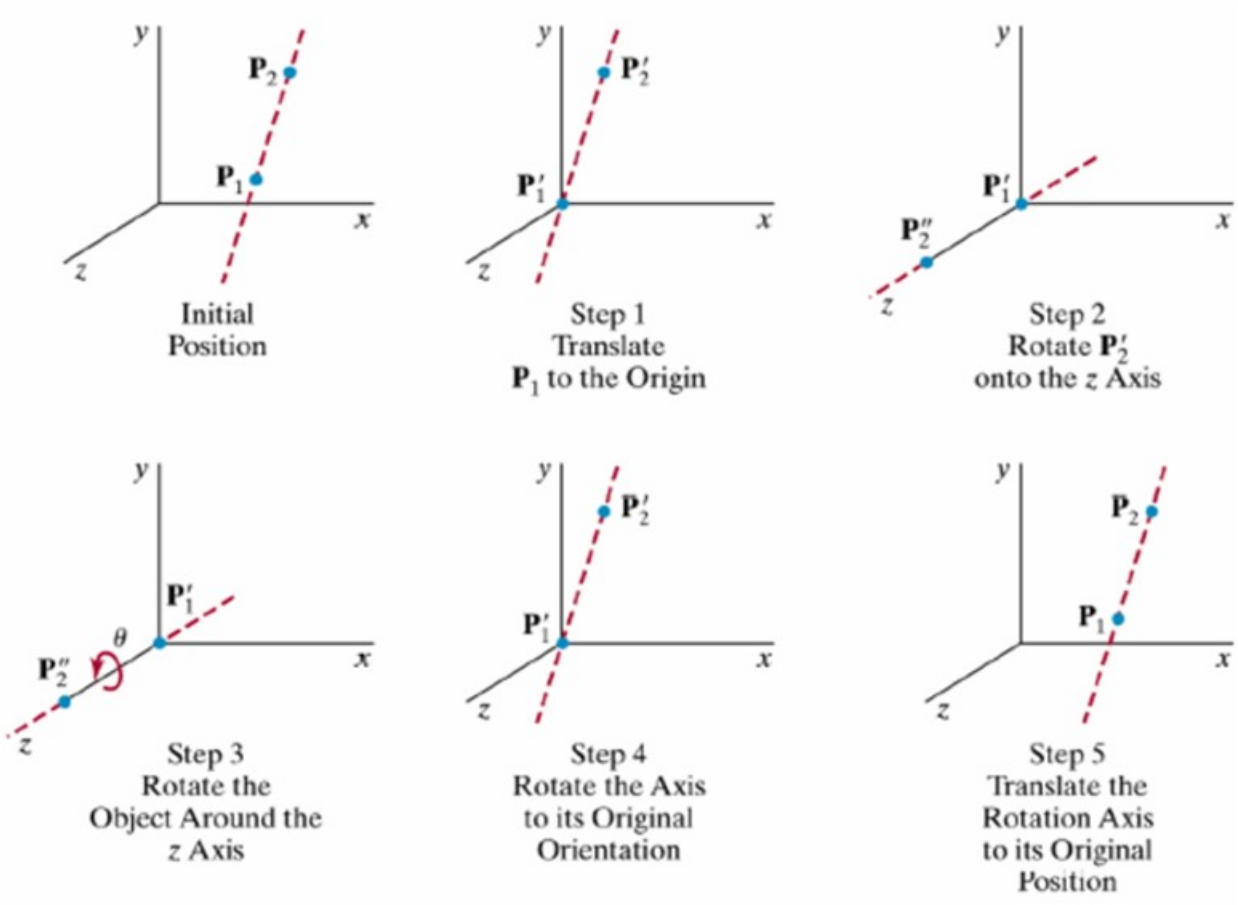
Transformations in 2D and 3D and also we used homogenised coordinates (don’t have to memorise the transformation matrices).

Composite transformations mean that we have two or more transformations performed on the same object.
We combine transformations of different type:

- translate, rotate, translate
- translate, scale, translate
- translate, reflect, translate

Matrix composition is not commutative. Be careful when applying a sequence of transformations.

Rotation around an arbitrary axis is an example to composite transformations.



- First translate the starting point P1 to the origin, now P1 is P1', and the end point P2 is P2'.
- Then rotate P2' onto the z axis. For this purpose, use unit vectors (use (P2-P1)/abs (P2-P1) to find the unit vector) and first rotate it around x axis to get the vector onto the xz plane, second, rotate vector u around y axis to get it aligned with z axis. So, there are two rotations here, with respect to x axis and y axis, respectively.
- Then rotate the object around z axis and rotate the axis to its original position.
- And lastly, translate this axis to it's original position.

The composite transformation matrix:

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x^{-1}(\alpha) \cdot \mathbf{R}_y^{-1}(\beta) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{T}$$

This order is important, from right to left, from first to last transformation respectively.
The rotated object coordinates can be found using dot product of this composite transformation matrix R and the matrix of the object's coordinates.

Similarly, scaling with reference to a fixed point requires first translation to origin, scaling with reference to origin and then translating back (the fixed point) to the original position. Dot product of these transformation matrices gives the composite transformation matrix.

Lighting and Shading

Light-material interactions cause each point to have a different color or shade in real world, so these lighting calculations are needed in CG.
Things that have to be considered in these calculations:

- Light sources,
- Material properties,
- Location of viewer,
- Surface orientation.

Since real light sources are difficult to model, in theory, simple light sources are defined.
First one is the **point source**.

The point light source has no shape and size, light is emitted from a single point.
The location of the light source is important for the calculations. The point light source is represented by position and color.
If we take this source to an infinite distance, instead of its location or distance to the object, its direction, that is, the direction of the light, becomes important (Sun light is an example). In this case, the result is a directional light source. It is considered as a point light source.
Second type, **spot light source**. We restrict the point light source angularly. Table lamp is an example. The rays illuminate up to a certain angle.
For **ambient** (the third type), there is equal light everywhere on the scene, on every object.
Intensity of the ambient light is the same at every point on the surface. Ambient light is also called background light.
Ambient light is actually used to model global effects. It refers to rays reflected from other surfaces and go to the object.
Point lights create sharp shadows but in reality, softer shadows are formed.
For this reason, **area light sources** were also modeled.
One step ahead of these are **extended light sources**, these are three-dimensional and require much more calculations.

The easiest illumination model for pipeline is the **basic or Phong model**. Its results are approximate.
In the basic illumination model, the amount of light reflected from a point is considered as a combination of three main components:

- Specular reflection
- Diffuse reflection
- Ambient light

These three component are linearly combined.

Diffuse component is for the amount of incoming light that is reflected equally in all directions.
Specular component is for the amount of light reflected in a mirror-like fashion.
Ambient sources have equal light on every object, everywhere on the scene. Ambient component is used to approximate light, arriving via other surfaces.

The light reflected from the surface of an object consists of ambient, diffuse and specular components.
Ambient component is independent of surface and camera position.
Diffuse is independent of the camera but depends on the angle of incidence of the light on the surface, so the position of the light source and the surface relative to each other is important.
Specular is like diffuse and the position of the camera is also important.

Clipping and Rasterization, Hidden Surface Removal

Clipping and rasterization (or scan conversion) are background processes.
Vertex inputs are first brought together after exiting the vertex operations in the pipeline and the primitives are created, assembled.
And then the clipping is applied, leaving a certain area of the scene to be displayed on the screen and discarding the other parts.
The following processes in the pipeline are not applied to the parts that are eliminated as the result of clipping.
Clipping determines which shapes or what parts of the shapes will be in the viewing area.
The shape to be clipped can be a point, a line (line segment), or a fill area (or a polygon). There are some algorithms to clip these shapes (formulas don't have to be memorised).

On a raster-scan system we draw with pixels, so, the pixels that are on or inside a primitive needs to be determined.
This task is figuring out which pixels to draw or fill on the screen, it is called rasterization (or scan conversion).
(Mostly talked about line drawing, formulas don't have to be memorised).

The final step of the pipeline is hidden surface removal or equally visible surface detection.
We want to see only the surfaces that are in front of other surfaces. There are some techniques for this purpose (just understand the logic, no need to memorise).