First two weeks (Introduction- Graphics Systems and Models- Graphics Programming)

What is CG?

An information presentation technology and deals with all aspects of rendering

Applications of CG

Information visualization (maps, building plans, …), Graphical user interfaces (GUI), Medical imaging (CT, MRI, PET, …), Computer Aided Design (CAD), Simulators, Games, Educational software, Animations, Virtual Reality – VR, Augmented Reality – AR, Mixed Reality – MR.

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) components.

Are used to be systems designed separately from computers but today PCs, workstations, game systems and mobile devices.

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.

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 (raster) are stored in the framebuffer.

What are the elements of image formation? (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).

What do we aim to do via CG?

We create synthetic/virtual objects with CG. We form 2D images. Both objects and the viewer exist in a 3D world, but the resulting image is 2D. To do so specify the positions of the geometric objects (points, lines, polygons, …), their vertices (plural form of vertex, a point in 3D space).

Pinhole and synthetic camera models

Please read how they work (from the slides, book or just google it).

How does a graphical application programmer interact with the graphics system (hardware)?

Through function specifications in a graphics library, application programming interface (API) like Direct3D and OpenGL.

How are the object geometries defined?

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

Right-handed vs. Left-handed Cartesian Coordinate Systems

positive x-axis points to the right, and the positive y-axis points up.
put either your left or right hand in the positive x direction.
direction your thumb points is the positive z-axis (for right-handed system toward you, for left-handed system away from you).
positive rotations are counterclockwise in right-handed, clockwise in left-handed.
Direct3D uses left-handed, OpenGL uses right-handed.

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 may 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.
(Please remember the figure in the related slide.)
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 primitice 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 on pixels. 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.

The third and the fourth weeks (Transformations)

What are coordinate systems?

Is a way of assigning numbers to points.

A number of different coordinate systems are there.

An important definition, frame of reference or reference frame: a coordinate system in which we are currently representing our objects.

Some coordinate systems:

Object or Modelling Coordinate System - An object can be specified using a coordinate system that makes its specification easy. It is a frame of reference from a specific object. Pick a point to be the origin of that object, and form the axes.

World Coordinate System - Base reference system for the overall model (including all of the objects). Different objects are brought together in this common system when setting up a scene. Its origin is at the center of the scene.

Camera (Eye or Viewpoint) Coordinate System - A frame of reference from our camera or eyes. We calculate everything with respect to the camera's or eyes reference frame.

Why do we need geometric transformations?

For change of coordinate systems (We have to do this change between different coordinate systems in CG, from object to world, from world to camera …). For computing object and camera movement in animations.

What are the geometric transformations?

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

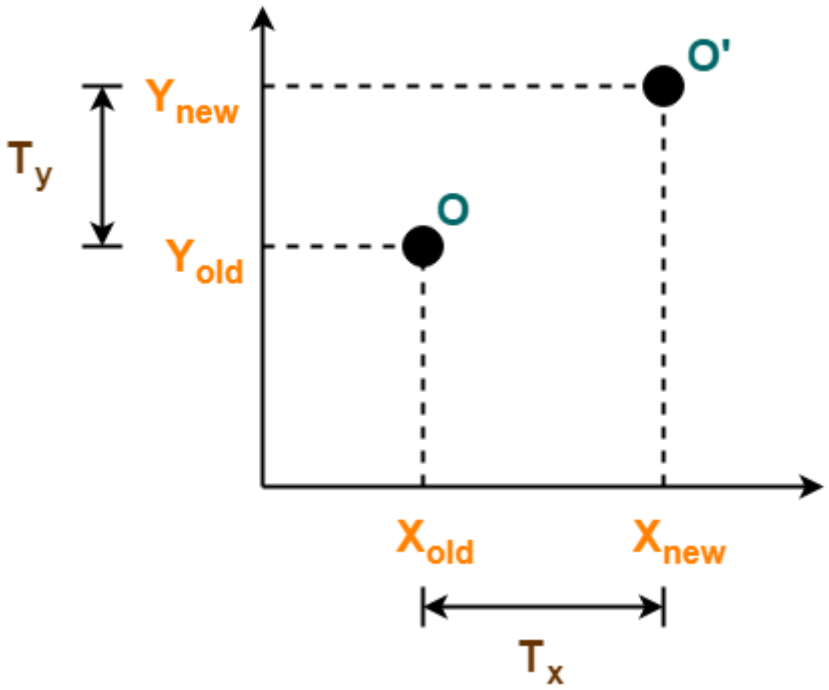
Rotation: moves the object. No deformation. Shape and size remain the same.

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.

T.1) Translation



Initial coordinates of the object O = (Xold, Yold)
New coordinates of the object O after translation = (Xnew, Ynew)
Translation vector or Shift vector = (Tx, Ty)

Tx defines the distance the Xold coordinate has to be moved.
Ty defines the distance the Yold coordinate has to be moved.

This translation is achieved by adding the translation coordinates to the old coordinates of the object:

Xnew = Xold + Tx (This denotes translation towards X axis)
Ynew = Yold + Ty (This denotes translation towards Y axis)

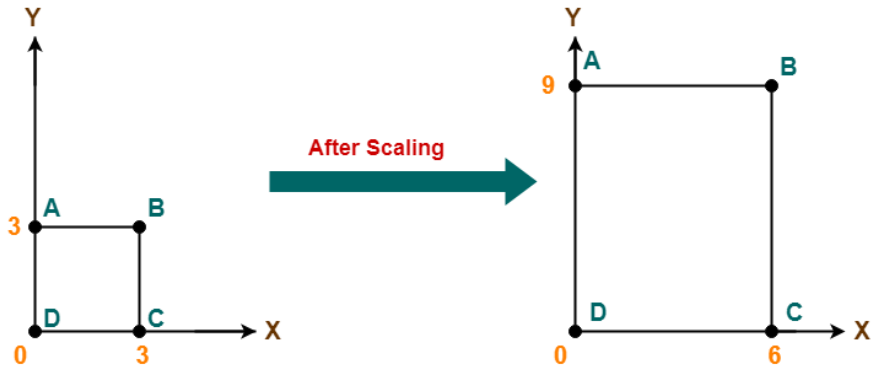
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

We used matrix addition to find the new coordinates.

T.2) Scaling

Initial coordinates of the object O = (Xold, Yold)
Scaling factor for X = Sx
Scaling factor for Y = Sy
New coordinates of the object O after scaling = (Xnew, Ynew)
This scaling is achieved by using these scaling equations:

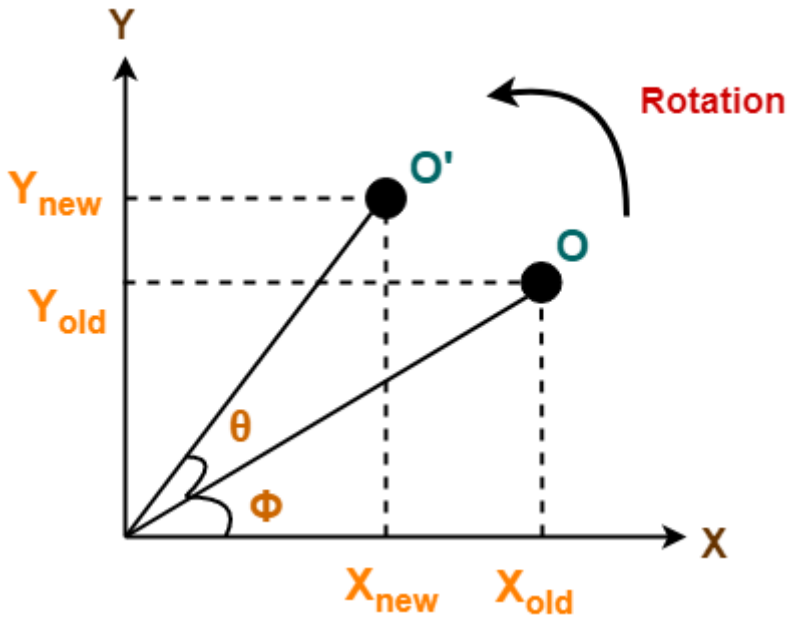
Xnew = Xold.Sx
Ynew = Yold.Sy



$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

We used matrix multiplication to find the new coordinates.

T.3) Rotation



Initial coordinates of the object O = (Xold, Yold)
Initial angle of the object O with respect to origin = Φ
Rotation angle = θ
New coordinates of the object O after rotation = (Xnew, Ynew)
This rotation is achieved by using these equations:

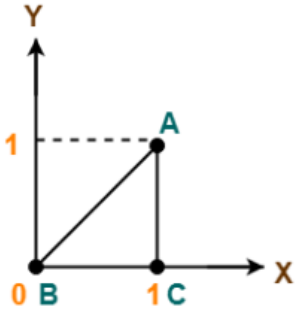
$$X_{new} = X_{old}.\cos\theta - Y_{old}.\sin\theta$$
$$Y_{new} = X_{old}.\sin\theta + Y_{old}.\cos\theta$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

We used matrix multiplication to find the new coordinates.

T.4) Shearing

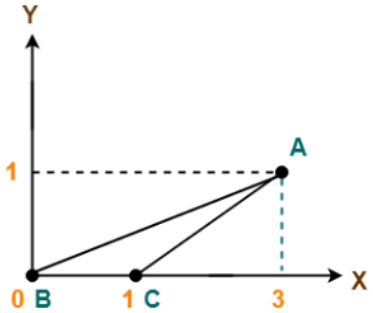
We will shear point O.
Initial coordinates of the object O = (Xold, Yold)
Shearing parameter towards X direction = Shx
Shearing parameter towards Y direction = Shy
New coordinates of the object O after shearing = (Xnew, Ynew)



	Shearing in X	**Shearing in Y**
Xnew	Shx . Yold + Xold	Xold
Ynew	Yold	Shy . Xold + Yold

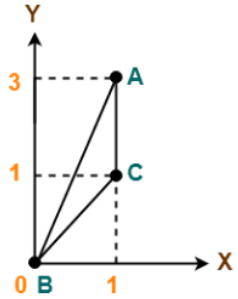
Shearing in X

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$



Shearing in Y

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \cdot \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$



We used matrix multiplication to find the new coordinates.
Note that double shearing (we want to shear the object two times) is not commutative, result depends on the order of shearings.

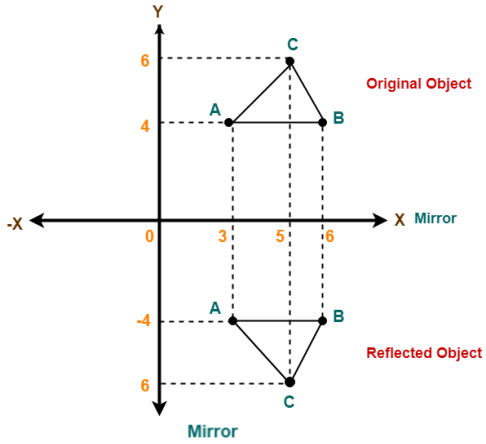
T.5) Reflection

Reflection is a kind of rotation where the angle of rotation is 180 degree.
The reflected object is always formed on the other side of mirror.
The size of reflected object is same as the size of original object.
We will reflect point O.
Initial coordinates of the object O = (Xold, Yold)
New coordinates of the reflected object O after reflection = (Xnew, Ynew)

	Reflecting on X	**Reflection on Y**
Xnew	Xold	- Xold
Ynew	- Yold	Yold

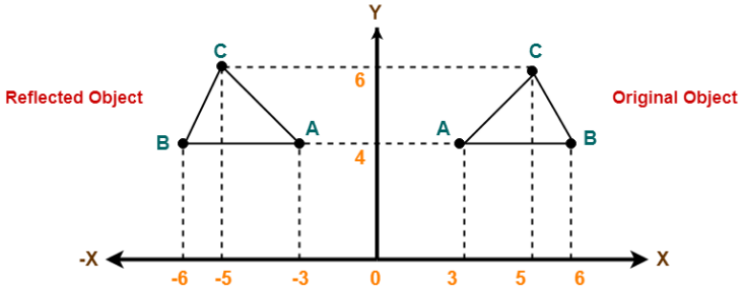
Reflecting on X

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$



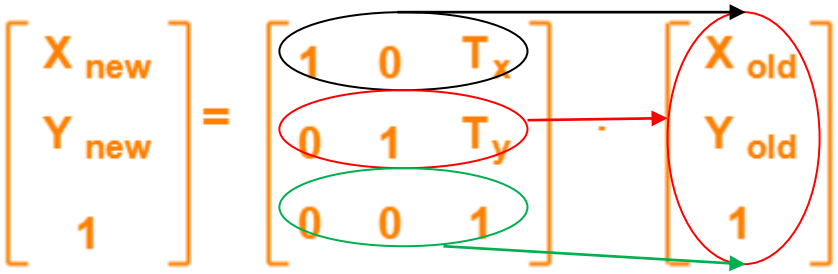
Reflecting on Y

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$



Again we used matrix multiplication to find the new coordinates.

Why do we need homogenous coordinates?
As we see, for **rotation, scaling, shearing, and reflection** we use matrix **multiplication**, but for **translation** we use **addition**.
We would like, all transformations to be multiplications, so we can concatenate them.
To do so, we need expressing points in **homogenous** coordinates. We add an extra coordinate, W (it's value is 1)



If we do this multiplication, we obtain:

Xnew = Xold + Tx (This denotes translation towards X axis)
Ynew = Yold + Ty (This denotes translation towards Y axis)
1=1

So, we now translate the point (or object) using multiplication instead of addition, via the homogenous coordinates.

Table of Transformation Matrices

	2D Coordinates	2D – Homogenised Coordinates	Explanation (2D)	3D – Homogenised Coordinates	Explanation (3D)
Translation	$\begin{bmatrix} T_x \\ T_y \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$	For 2D matrix, addition is applied. Distances are in the 3rd column in homogenised coordinates (multiplication is applied)	$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
Scaling	$\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$	$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Both 2D and 2D homogenised are the same, but homogenised one has one more coordinate, it is 3x3 matrix Done with reference to origin in 2D	$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
Rotation	$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$	$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$		$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Rotation around z axes, it is like rotating with respect to origin in 2D case

	2D Coordinates	2D – Homogenised Coordinates	Explanation (2D)	3D – Homogenised Coordinates	Explanation (3D)
Shearing in X	$\begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	X coordinate changes depending on Y	$\begin{bmatrix} 1 & Sh^y_x & Sh^z_x & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	X coordinate changes depending on Y and Z
Shearing in Y	$\begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Y coordinate changes depending on X	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ Sh^x_y & 1 & Sh^z_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Y coordinate changes depending on X and Z
Shearing in Z	-	-		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ Sh^x_z & Sh^y_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Z coordinate changes depending on X and Y
Reflection on X (3D: relative to YZ plane)	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
Reflection on Y (3D: relative to XZ plane)	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

	2D Coordinates	2D – Homogenised Coordinates	Explanation (2D)	3D – Homogenised Coordinates	Explanation (3D)
Reflection on Z (3D: relative to XY plane)	-	-		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

Transformation matrix of 3D rotation is for rotation around z axes, it is like rotating with respect to origin in 2D case.
For rotating around x, use:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For rotating around y, use:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

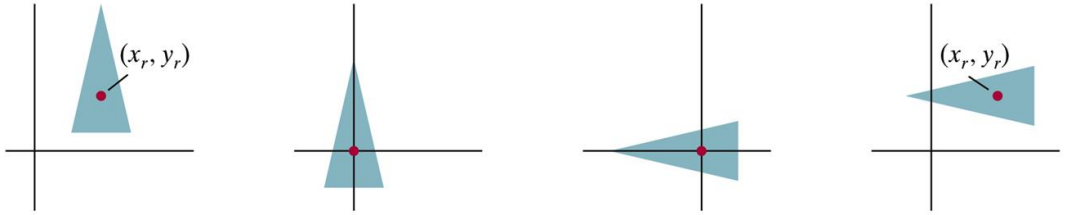
Note that, column and the row of the axis that we rotate the object around, does not include angle values.

Composite transformations

Two or more transformations performed on the same object.

We combine transformations of different type.

Example to 2D composite transformations



(a) Original Position of Object and Pivot Point

(b) Translation of Object so that Pivot Point (x_r, y_r) is at Origin

(c) Rotation about Origin

(d) Translation of Object so that the Pivot Point is Returned to Position (x_r, y_r)

We want to rotate an object **with respect to a specified pivot point** x_r, y_r. In 2D, we should do rotation **with respect to origin**.

1. Translate the object (pivot point is then at the origin), $-x_r$, $-y_r$ are distances.
2. Rotate the object with reference to the origin.
3. Translate it back (pivot point is then returned to it's position), x_r, y_r are distances.

So, the composite transformation matrix is production of these three matrices.

$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) =$$

$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

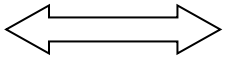
* We should write these three transformation matrices from right to left (not commutative)

* CTM (Coordinate Transformation Matrix):
Translation to origin T (-,-) . Rotation R . Translation back T (+,+)

Actually we can do these transformations one-by-one also, it is the same.

Assume that this is a point P: $\begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$

P1 = T(-,-) . P
P2 = R . P1
P3 = T (+,+) . P2



P3 = CTM . P

* An object transformation alters the coordinates of each point on the object according to some rule, leaving the underlying coordinate system fixed.

* A coordinate transformation defines a new coordinate system in terms of the old one and then represents all of the object's points in this new system.

Please study the topics below:

- properties of transformations
- transformations between coordinate systems
- rotation around an arbitrary axis & scaling w.r.t. a fixed point
- alternative method of rotation & vector bases