Computer Graphics

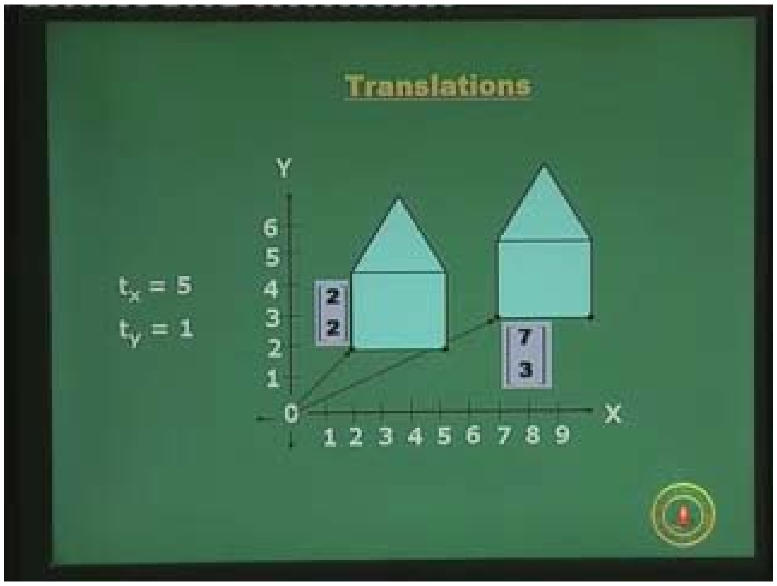
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 7
Transformations in 2-D

Welcome everybody. We continue the discussion on 2D transformations or two dimensional transformations. In the last lecture we have been discussing the various forms of the transformation matrix T basically the elements and we have almost known the different combinations of the values of the elements that create different types of transformations and we have been mainly discussing four types of transformations namely rotation, scaling, shear and reflection.

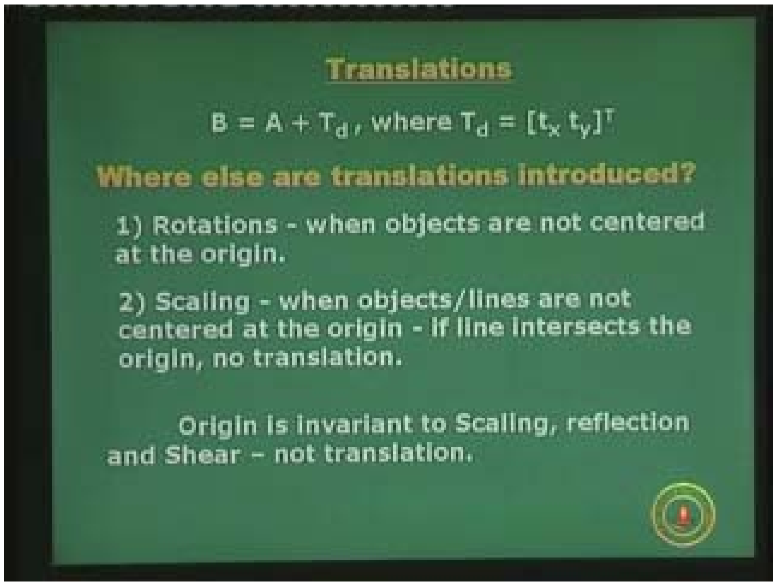
Rotation, scaling, shear, reflection are the four different types and we know that the combination of the diagonal elements create certain types of transformation. Off-diagonal creates one, rotational matrices and again orthogonal, we know all those properties as we have seen in the previous lecture. And also, I did mention that for some reason I did not discuss the very simple fundamental transformation as the translation. Translation was not discussed and it is the reason why you will know soon. Well, let us look into the slide and look at an example of translation. As you see here, translation is a movement of one object to another. In the slide there is an object located.

(Refer Slide Time: 00:02:23)



The coordinates of just one pixel value on the left bottom most corner of the structure is given which is 2, 2 on the left side here and you add 5 to 2 you get 7 and you add 1 to 2 you get 3. So 2, 2 is displaced to 7, 3 and all other coordinates are also displaced in a similar manner because we are talking of rigid body transformation so all of these points or the entire object undergoes the same types of transformation like a translation. If you want to replace the translations by matrix equation you can easily represent it by an equation like this. You remember the coordinates of A and B are 2 into 1, basically it is a vertical column vector with just two elements and you add t_x to X and t_y to Y and you get the coordinates of the displaced point or for the entire object.

(Refer Slide Time: 00:04:37)



Translations

$B = A + T_d$, where $T_d = [t_x\ t_y]^T$

Where else are translations introduced?

1) Rotations - when objects are not centered at the origin.

2) Scaling - when objects/lines are not centered at the origin - if line intersects the origin, no translation.

Origin is invariant to Scaling, reflection and Shear – not translation.

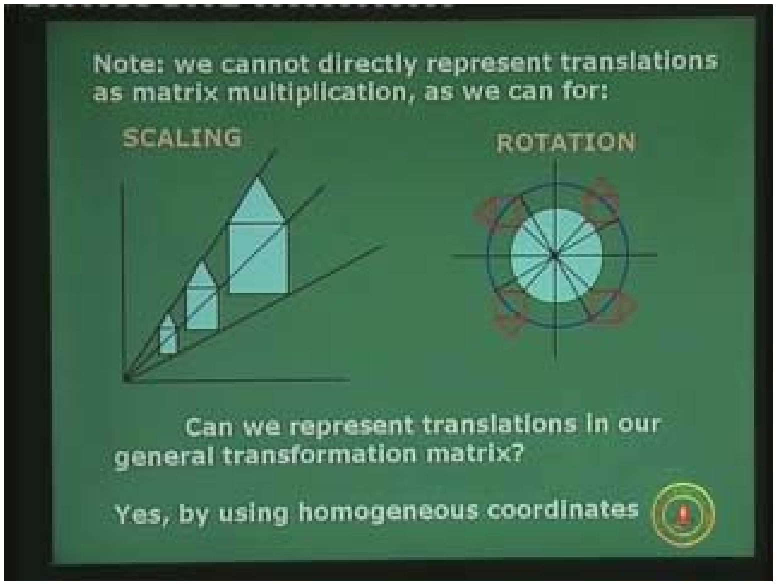
And translations basically too happen in almost all sorts of transformations. It is done inherently. If you do not write it even as an expression and do not give it a translation also it is there in other types of transformation as for example rotation.

Rotations of any objects which are not centered at the origin undergo a translation and are the same for scaling when objects and lines are not centered at the origin. Anything which is not at the origin and undergoes rotation and scaling then those objects including lines and points also undergo a translation in some form.

We have seen examples of rotation earlier. We will again see some examples today and it will be clear to you that translation inherently happens with most other types of transformations as well. But of course if the object or the point is at that origin or the line intersects the origin there may not be a translation. When you give it a rotation or a scaling there may not exist any such thing.

Or we will say that the origin by itself is invariant to scaling reflection and shear but it is not invariant to translation that is indeed a very nice observation because if you plug on the top of the expression given in the top of the slide here and substitute A is equal to 0 which is the origin or the coordinates of the origin into the space you get B equals T_d which is t_x and t_y. So the origin is shifted based on the coordinates or the amount of displacement vector. Basically the translation components might shift, you should be able to shift something to origin and also the origin to somewhere else. But of course you cannot scale an origin, you cannot reflect it and you cannot shear it, there is the key.

(Refer Slide Time: 00:06:12)


Note: we cannot directly represent translations as matrix multiplication, as we can for:

SCALING
ROTATION

Can we represent translations in our general transformation matrix?

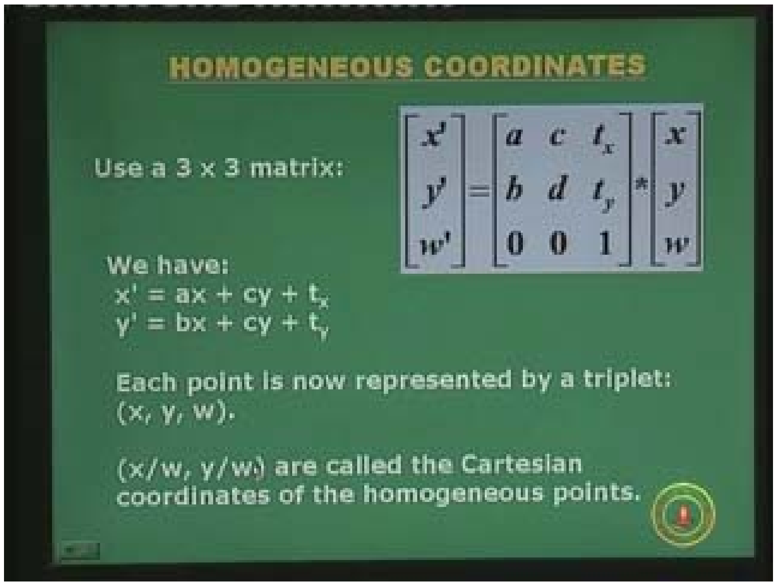
Yes, by using homogeneous coordinates

So, you see the expression which we talked about in the last class in terms of the transformation. We used matrix multiplication in general to represent all transformations in 2D and except translation we have an addition instead of a multiplication.

So we have a mathematical problem, we cannot directly represent translations as matrix multiplication as we can do for say, these are examples in figures of scaling and rotation. Again in the same structure like a house which is scaled up or scaled down it means it is expanded or contracted or we can make something like a house to rotate about its origin. And in this case they may inherent a translation but when you represent these transformations you use the matrix multiplication operation, mathematical model. But for translation I have given in the previous slide we are forced to use an addition.

How can we combine these two? Because its not easy to do that, so how can we represent translations in our general transformation matrix which is a b c d four parameters, where do we plug in the transformation elements for the translation and its made possible with the help of what are called the homogeneous coordinates. We will see what are the homogeneous coordinates are the representation of a point in a homogeneous coordinate system.

(Refer Slide Time: 00:07:23)



HOMOGENEOUS COORDINATES

Use a 3 x 3 matrix:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & c & t_x \\ b & d & t_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

We have:
$x' = ax + cy + t_x$
$y' = bx + cy + t_y$

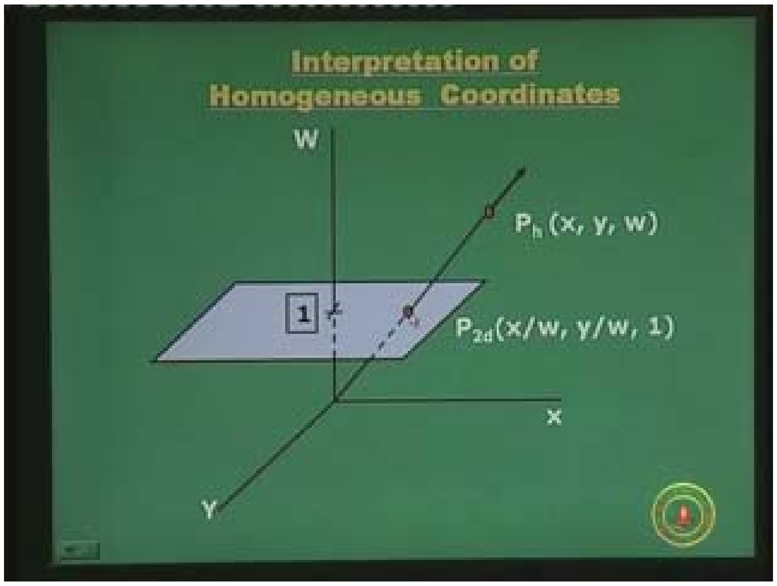
Each point is now represented by a triplet: (x, y, w).

(x/w, y/w) are called the Cartesian coordinates of the homogeneous points.

We will first see the 3 into 3 matrix or the transformation matrix in homogeneous representation but the homogeneous coordinates are there now to the right of you. Instead of x y we have a third element called w. In some books we used h instead of w, so you can use it interchangeably but stick to one again h or w. And x y w are the original coordinates of a point before transformation, it is transformed by a 3 into 3 transformation matrix and you get x prime y prime and w prime. What is the significance of w and what is the role? Before going to that, if you just take out the two scalar linear equation out of this matrix equation, you get x prime is ax plus cy plus t_x and of course y prime is similarly bx plus cy plus t_y.

What is this homogeneous coordinate? Each point is now basically represented by a triplet x y w. So there is a third element which comes, remember it is not a 3D space, we are still talking of 2D space, x y are the coordinates value and what is the role of w?
That is the homogeneous term, it is just a parameter and we will say once given x y w you divide the first 2x, y by the third element w or the third parameter w and you get back your Cartesian 2D coordinate and that is what the last line says. x by w y by w, how do you get this? Take the first two divide by third. And those are called the Cartesian coordinates of the homogeneous points. Therefore, homogeneous points coordinates are this; x y w or x y w here or in the equation on the right hand side x y w or x prime y prime w prime. But to get the Cartesian coordinates take the first two divide by third and you get x by w and y by w where you get the actual coordinates which we have been talking about for w could be equal to 1.

(Refer Slide Time: 00:08:21)

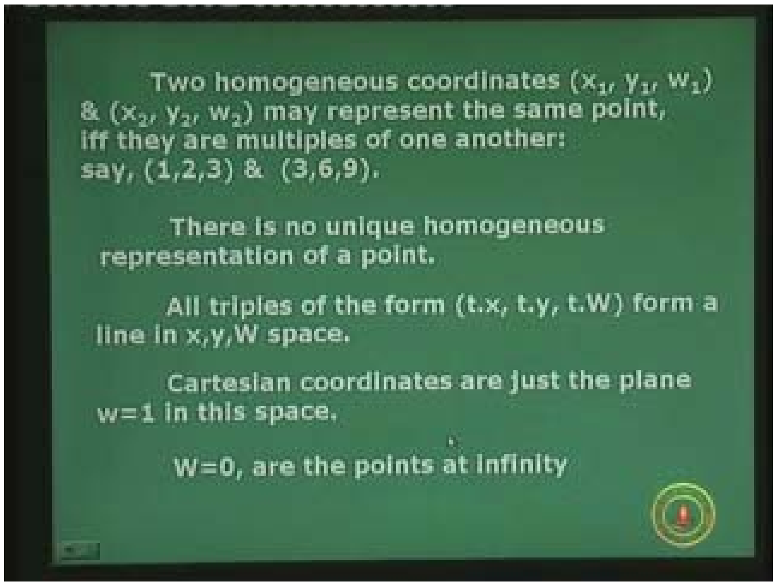


This is an interpretation of homogeneous coordinate representation. Any point in this vector line which comes out of this x y w space, you can visualize this now to be a 3D space where w could be z but actually it is not z because x y are the coordinates and any vector which points in this direction x y w. Ph is a point in this homogeneous space or in x y w space and it also points as a vector along a certain direction. Any point on this line or on this vector in this direction basically represents a single point in Cartesian coordinate system which is this one.

So the P_{2d} is nothing but x by w y by w and one and that is obtain by dividing all the x y w triplets by the third element w. And that is what gives you, the first two of those gives you the 2D coordinates. So basically this intersection of this vector with w equal to one plane that is you draw a plane which is parallel to x, y axis and normal or orthogonal to the w axis, the w axis is perpendicular and normal to this plane and it intersects the w axis at w equal to 1 which is marked here at the center.

Again I repeat. This plane which is parallel to x y plane, perpendicular to w, y axis intersects the w axis equal to 1. This plane intersects the corresponding vector, this Ph vector at a point which will give you the 2D coordinates of the actual point in 2D space. So this is the interpretation of homogeneous coordinates.

(Refer Slide Time: 00:11:47)



Two homogeneous coordinates (x_1, y_1, w_1) & (x_2, y_2, w_2) may represent the same point, iff they are multiples of one another: say, (1,2,3) & (3,6,9).

There is no unique homogeneous representation of a point.

All triples of the form (t.x, t.y, t.W) form a line in x,y,W space.

Cartesian coordinates are just the plane w=1 in this space.

W=0, are the points at infinity

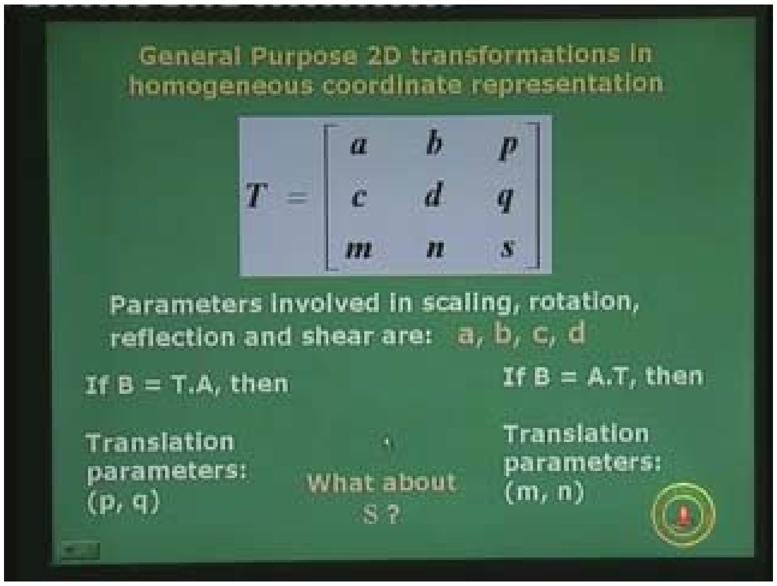
More examples on this, if you take any two homogeneous coordinate x_1, y_1, w_1 and x_2, y_2, w_2 they may actually represent the same point if and only if they are multiples of one and other. A typical example is 1, 2, 3 and 3, 6, 9, if you see here I did say that you can obtain the 2D coordinates from the homogeneous form by dividing the first two elements by the third. So 1 by 2, 1 by 3 and 2 by 3 are the respective Cartesian coordinates. You can get the same thing here because 3 by 9 is 1 by 3, 6 by 9 is also 2 by 3. Or you can multiply all the values by a constant scalar quantity 3 and you get the other one. So basically you can scale down or scale up the same point. There is no unique homogeneous representation of a point.

This 1, 2, 3 is one point which is equivalent to 1 by 3, 2 by 3 in Cartesian coordinate system which is the same as 3, 6, 9. Or I can take it to be 6, 12, 18 or I can take it to be 100, 200, 300 multiply this 1, 2, 3 by all the elements by the same quantity k and all those basically represents the same point. You can start to visualize the figure which I showed you in the previous slide. Their intersection of that vector with that plane, any point in that line of that vector can be represented by values and the intersection gives a unique point.

All triplets of the form t dot x, t dot y and t dot w form a line in the x y w space, we discussed about this vector earlier. And of course you get that by dividing w t dot w if you divide you get x by w y by w from this triplet. And the Cartesian coordinates are just the plane w equal to 1 in this space which we have seen in the previous slide. I am saying that again, so that is how you get the Cartesian coordinates from the homogeneous coordinates by intersecting this vector with the w equal to one plane or dividing the first two by the third element.

Well, what about w equal to 0, w should be a non-zero quantity because when we are talking of w equal to 0 we are talking about points at infinity in 2D space. We do not talk about those in real numbers or whole numbers which can be represented and stored in computer infinity; it cannot be stored so we do not need to bother about that but just the physical interpretation w equal to 0 represents points at infinity.

(Refer Slide Time: 00:15:10)



General Purpose 2D transformations in homogeneous coordinate representation

$$T = \begin{bmatrix} a & b & p \\ c & d & q \\ m & n & s \end{bmatrix}$$

Parameters involved in scaling, rotation, reflection and shear are: a, b, c, d

If B = T.A, then

Translation parameters: (p, q)

What about S ?

If B = A.T, then

Translation parameters: (m, n)

Once we know what are the homogeneous coordinates and its representation we can now re-look at the general purpose 2D transformation matrix in general which should be able to handle all different five types of transformations and this is a general purpose transformation matrix. So there are 9 elements, 3 into 3 matrix, we need a 3 into 3 because homogeneous Cartesian coordinate representation talks about a 3 into 1 element column vector. So I need a 3 into 3 element transformation matrix.

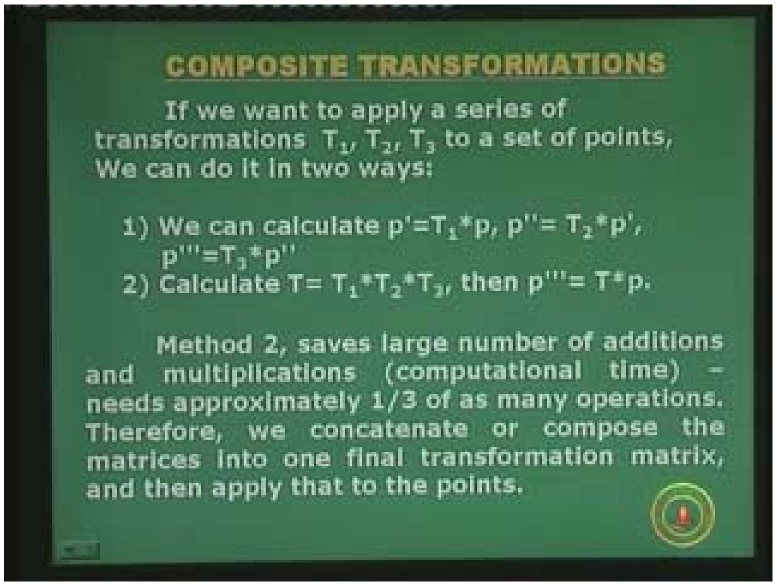
The sub-matrix consisting of the top left four elements a b c d are the parameters involved in scaling rotation reflection and shear as for the previous case though that part is unaltered. We still have a b c d with the same significance as we have been talking in the previous lecture with the four different types of transformations such as scaling, rotation, reflection and shear. So there is absolutely no problem with a b c d because those corresponding four parameters behave in the same way. We still have five different parameters out of those which two represent translation is most important to us. We need to know what is translation.

Now I talked about two different notations; a premultiplication and post multiplication operator to obtain the new coordinates B from A. And if you are premultiplying the coordinates A by T the translation parameters are p, q as given here at the bottom. So, in the third column, the first two indicate translation if you are premultiplying and if you are post multiplying A by T you have m, n the first two elements of the last row of the matrix T indicate transformation.

We talked about this pre and post scenarios, row a becomes either a row or a column vector depending upon whether you are premultiplying or post multiplying and depending on that the corresponding translation parameters T_x and T_y do change place in fact. There could be either p q or m, n depending upon whether using B equals T multiplied by A or A multiplied by T. Of course you will have another question that if m, n is a translation parameter what happens to p, q and in p, q is a translation and what happens to m and n? We will hold on that for the time being but do not worry about it right now.

But we still have a ninth parameter which is the s. Can you almost guess what s could be doing? Well I leave it as an exercise for you to try out the right equations by keeping s as non 0 and a and d to be 1 and all other elements, that means basically take this as an identity matrix with s equal to 2 and all other elements c m and n p q b these are all off-diagonal elements to be 0 a and d diagonal elements to be 1 and s to be 2 or 3. I am sure if you do not make a mistake you will find that the s also gets involved in uniform scaling. Coming back to composite transformations: Because now we know to handle all the five different transformations that means reflection, rotation, scale, shear and translation.

(Refer Slide Time: 00:17:54)



COMPOSITE TRANSFORMATIONS

If we want to apply a series of transformations T_1, T_2, T_3 to a set of points, We can do it in two ways:

1) We can calculate $p'=T_1*p$, $p''=T_2*p'$, $p'''=T_3*p''$
2) Calculate $T=T_1*T_2*T_3$, then $p'''=T*p$.

Method 2, saves large number of additions and multiplications (computational time) – needs approximately 1/3 of as many operations. Therefore, we concatenate or compose the matrices into one final transformation matrix, and then apply that to the points.

Of course we did not discuss necessarily in this order but there are five different transformations which are possible by manipulating the different elements of the transformation matrix.

And what happens now, when we want to apply two different transformations either together or one after the other, that is known as composite transformations which states that if you want to apply a series of transformations T_1 then T_2 and then T_3 not in some sense together but in a sequence where the sequence is very important. We will come to that in a moment from now, but you have these three transformations applying and let us see them one after the other. The effect is the same as if it is applied together, may be and it is applied to a set of points or a single point which we can do in two ways.

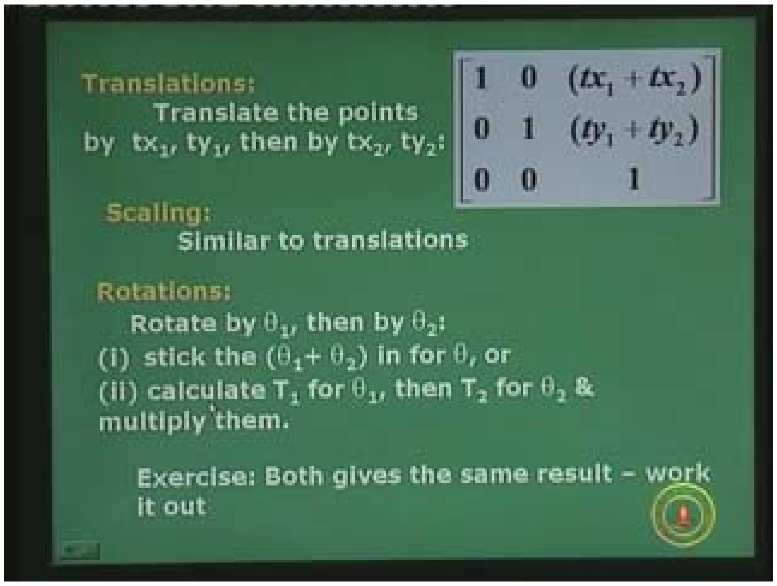
Number one; we can calculate from the vector p which is the coordinates of the point p. Of course we are discussing homogeneous coordinate representation and the T is also a 3 into 3 transformation matrix which could represent any transformation. And in this case we are taking premultiplication of p by T. So we multiply p by T_1 and you get p prime, then from p prime you can premultiply by T_2 to get p double prime which means that you are first applying the transformation of T_1 on p to get p prime. Then you apply transformation T_2 on p prime to get p double prime and ultimately on p double prime you apply a transformation T_3. You should be able to get p triple prime or basically the resultant coordinates of the point p after three successive transformations.

You can do three different multiplications as such, done one after the other or you can do the other way. You can calculate T as a product of all the three transformations and then apply and then obtain p prime as T multiplied by p where T is given by this. This you can easily derive from the number one step.

It starts substituting p double prime here and then p prime there you will find that the T is nothing but a sequence of matrix multiplication of T_1, T_2, T_3 and then you get p prime by a single matrix multiplication. Actually in most scenarios you have this multiplication matrix form known. And if it is actually known you do not have to compute for a given problem the resultant matrix T for T transform. So in fact you are saving a lot of time in this process and the method too is optimal. That means it saves a large number of additions and multiplications and ahead gives you low computational cost or low computational time. The computational complexity of the second step operation becomes really low.

The computational complexity is really low as it needs approximately one third of as many operations as in step number one. Therefore we concatenate or compose the matrices that is you get T_1, T_2, T_3 together and put it to one final transformation matrix T and then apply that to all the points or a to single point to get the final transformed point.

(Refer Slide Time: 00:20:23)



Translations:
Translate the points by tx_1, ty_1, then by tx_2, ty_2:

$$\begin{bmatrix} 1 & 0 & (tx_1 + tx_2) \\ 0 & 1 & (ty_1 + ty_2) \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling:
Similar to translations

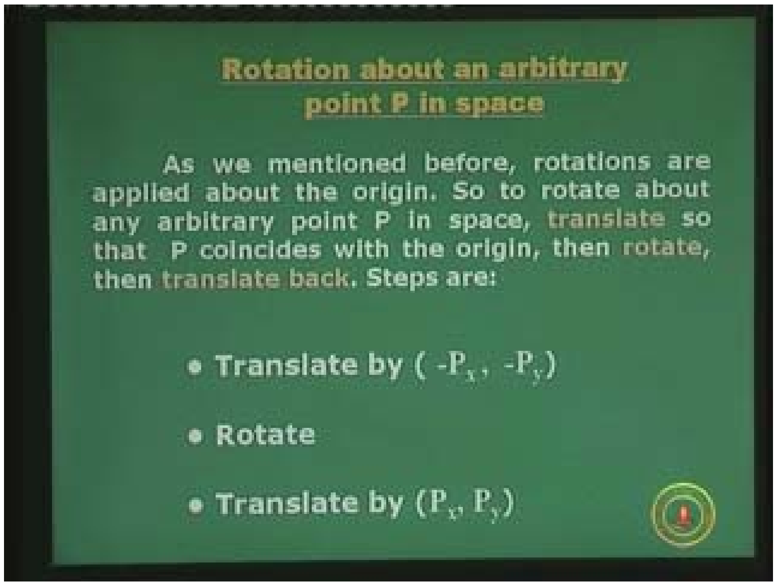
Rotations:
Rotate by θ_1, then by θ_2:
(i) stick the $(\theta_1 + \theta_2)$ in for θ, or
(ii) calculate T_1 for θ_1, then T_2 for θ_2 & multiply them.

Exercise: Both gives the same result – work it out

Just to take few simple examples I was talking if you want to apply two translations one after the another that means you need to translate the points by tx_1, ty_1, x and y indicate the translation components along x and y direction respectively and then you want to translate it by again by tx_2 to ty_2. I leave it as an exercise for you find out that the overall translational matrix which is a multiplication of the two translational components we are taken premultiplication in this case is given in the right hand side which is very simple. The scaling effect is also similar to that of translations and these scaling coefficients will come along the diagonal elements of the matrix T.

Rotation is a little bit tricky but there are good methods to solve it. The problem says that you need to rotate by an amount or an angle theta 1 and then you need to rotate it by an angle theta 2. Instead of applying two different transformations such as 1 for theta one then for theta two in succession, you put in the transformation matrix for rotation an angle theta 1 plus theta 2 itself in place of theta. In the original transformation matrix where you remember, you had a cos theta minus sin theta, sin theta cos theta instead of replacing the theta by theta one plus theta two and then you better do that instead of doing the second stage which will be costly, that means calculate T_1 for theta_1, T_1 to theta_2 and then multiply. If you multiply them the effect which you get basically is the same by replacing theta by theta 1 plus theta 2. Take it as an exercise that both step one and step two gives the same results of the rotational matrix.

(Refer Slide Time: 00: 22:17)



Rotation about an arbitrary point P in space

As we mentioned before, rotations are applied about the origin. So to rotate about any arbitrary point P in space, translate so that P coincides with the origin, then rotate, then translate back. Steps are:

- Translate by ($-P_x$, $-P_y$)
- Rotate
- Translate by (P_x, P_y)

We will come into that form. Often you need to rotate about an arbitrary point in space, because if you think of the rotational matrix which we discussed in the previous lecture of transformation it was basically rotation with respect to the origin that means you are rotating with respect to the origin. So we have the transformation matrix which helps you to rotate about the origin. What about trying to rotate about any arbitrary point of space, what do you mean by that?

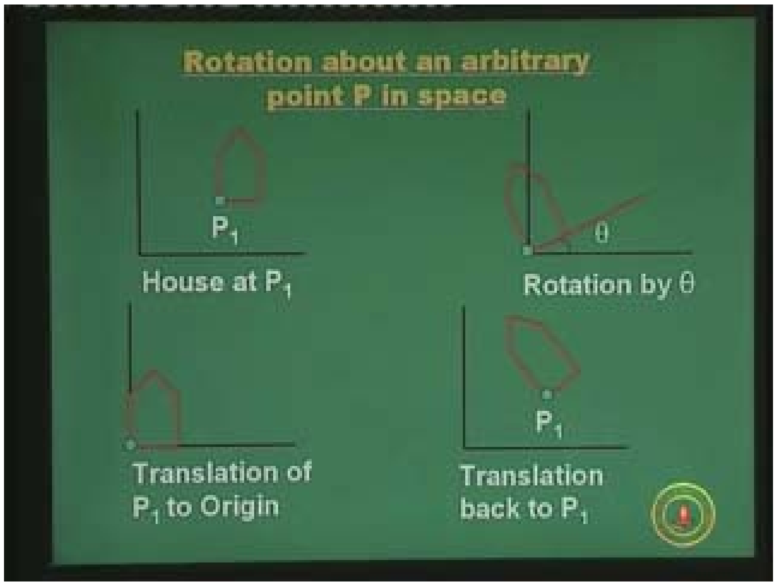
Let us take the revolution or rotation of the earth. The earth spins about in its own axis. So you can apply the transformation matrix for rotation which you know. If you are applying the rotation about the center point about the earth then it is fine. But let us take the case of the moon. So the points which are rotating on the moon about the center of the moon, of course the moon revolves around the earth like the earth also moves around the sun but if you are if you are trying to move an object at a point with respect to some other point which is not the origin we do not have the transformation matrix and we will derive it right now.

What is the first step?
As we mentioned above rotations are about the origin. Look into the slide you will find out. As you have mentioned before rotations are applied about the origin, so to rotate about any arbitrary point P in space there are two steps translates so that P coincides with the origin. As you see in the slide here there is a new paragraph. So to rotate about any arbitrary point P in space translates so that P coincides with the origin and that is the first step.

The second step is rotate, we can rotate now because the point has become the origin or the origin has become the point, whatever way you look at it and then you can translate it back. So these steps to implement the overall transformation are translate by minus P_x, P_y which are nothing but the x coordinates of the point P and y coordinates of the point P.

So you are basically translating the point P to the origin or the origin moves to the point P, whatever way you look at it. Then you rotate using the original rotation matrix and then finally translate it back. Let us have an illustration of this step, of the process first and then what we will do is look at the overall matrix of the transformation.

(Refer Slide Time: 00:24:14)



Rotation about an arbitrary point P in space

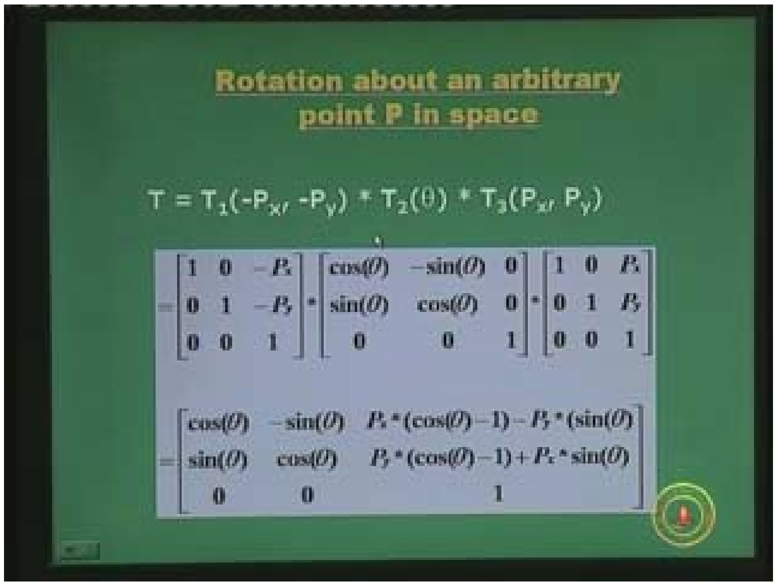
So rotation about an arbitrary point P in space; we are discussing this point. I will take the example of the house without the shading and I want to rotate this house with respect to the point P_1 which is at the bottom left of the screen and not with respect to the origin. If I want with respect to the origin which is here there is no problem because I have the rotation matrix for that and the whole object along with point P_1 should rotate.

Basically when I rotate with respect to P_1, the point P_1 will stay and everything else will rotate around that. So that is the difference between rotating about a point, arbitrary point in space and in this case arbitrary point is on the object or with respect to the origin. If I rotate about the origin I repeat, the entire object will rotate along with all the points but in this case with respect to point P_1. The point P_1 we will see, it is there and the other points will be changed. So what was the first step if you recollect?
Translate it to the origin, translate P_1 to the origin by translation vector minus P_x minus P_y and we have seen that.

Now we can rotate it because P_1 is at the origin. So after rotation by an amount theta you will find that it is equivalent to transforming the two dimensional plane itself and I have drawn a red line as the x axis just to show you, but basically you are rotating only the house and not the other axis. And finally translate it back to P_1. You will compare the first figure on the top left and the final figure form top right you can see that the point P_1 has not changed its position, the structure has basically rotated by an amount theta. So this is an example of a sequence of the different stages or the steps involved in rotation about an arbitrary point P in space.

If you wanted to rotate the structure about this origin actually what would have happened with respect to theta is that the structure would have almost gone past the y axis vertical axis or would have at least gone near to that. So I leave that as an exercise for you to rotate the structure about this origin. You will require a simple rotational matrix which should do that.

(Refer Slide Time: 00:25:17)



Rotation about an arbitrary point P in space

$$T = T_1(-P_x, -P_y) * T_2(\theta) * T_3(P_x, P_y)$$

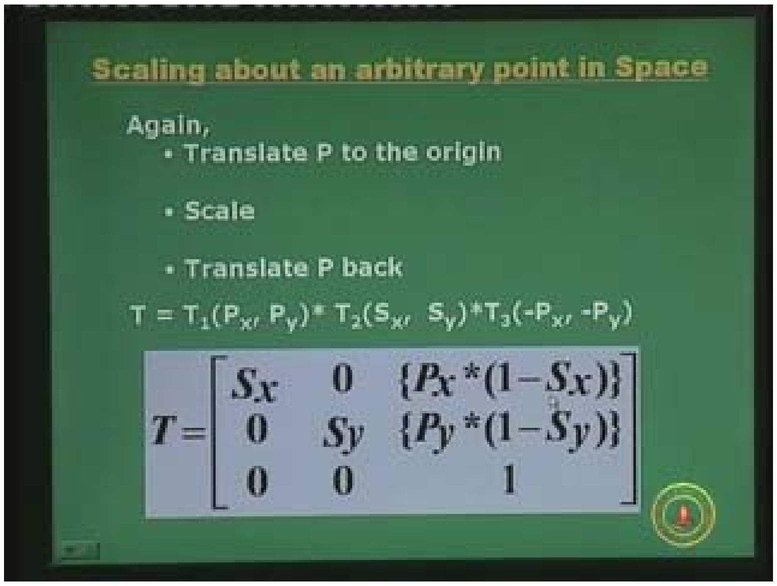
$$= \begin{bmatrix} 1 & 0 & -P_x \\ 0 & 1 & -P_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & P_x \\ 0 & 1 & P_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta) & -\sin(\theta) & P_x * (\cos(\theta) - 1) - P_y * (\sin(\theta) \\ \sin(\theta) & \cos(\theta) & P_y * (\cos(\theta) - 1) + P_x * \sin(\theta) \\ 0 & 0 & 1 \end{bmatrix}$$

Let us come back to the equations. We had seen that there are three stages, so you have a translation matrix, then a rotational matrix T_2 and then a translation matrix T_3. All these are general transformation matrices 3/3 representing homogeneous Cartesian coordinates form and the corresponding matrices are given here.

So the T_1 with minus P_x P_y is given on the left hand side multiplied with the, this is the original rotational matrix with respect to a point at origin and cos theta minus sin theta and so on and all are 3/3 matrix finally translating the point back. If you multiply all these three matrices, I leave it as an exercise for you to find out whether you can actually multiply by starting with the last two or the first two as it does not matter however you start but if you are correct you will get back this final expression. So what you need is almost the system or a program used to remember these expressions of the elements of this for a general purpose rotation. P_x P_y are the coordinates of the point about which you want to rotate and theta is the amount of rotation which is the final form of the rotation of an object or a point about any arbitrary point P in space. That is the example of a composite transformation that means several transformations together.

(Refer Slide Time: 00:26:55)



Scaling about an arbitrary point in Space

Again,

- Translate P to the origin
- Scale
- Translate P back

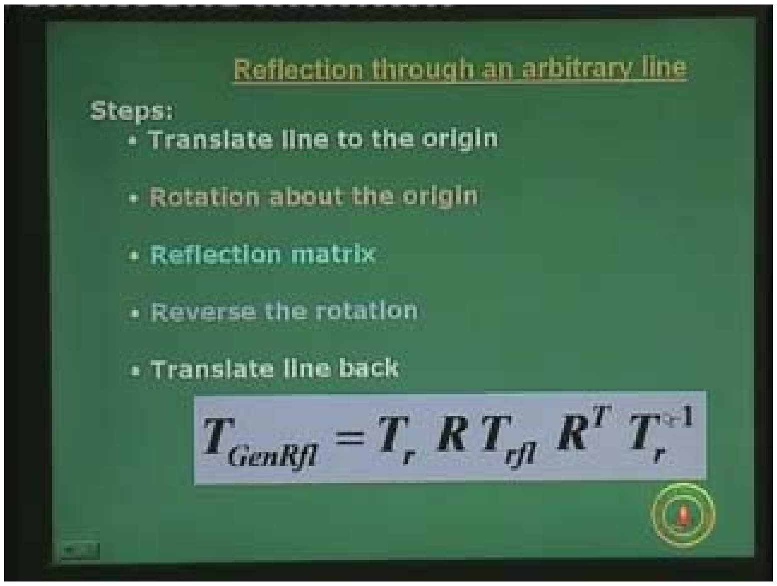
$T = T_1(P_x, P_y) * T_2(S_x, S_y) * T_3(-P_x, -P_y)$

$$T = \begin{bmatrix} Sx & 0 & \{Px*(1-Sx)\} \\ 0 & Sy & \{Py*(1-Sy)\} \\ 0 & 0 & 1 \end{bmatrix}$$

Well, you can also talk about scaling about an arbitrary point in space similarly. And since we have transformation matrices which can scale it with respect to the origin, the similar concepts as we did for rotation is applicable here, that means you translate P to the origin scale and translate it back. So you look here, these are the three stages translate P to origin scale, translate P back and three transformation matrices S_x S_y at the diagonal elements. For x non-uniform scaling could be S_x not equal to S_y but this is a diagonal term and we also have the translational parameters at the T_1 and T_3.

I leave it as an exercise for you to checkout that the overall transformation matrix T after multiplications of these transformation matrices will give you an expression which is given in the figure below. I would want you to note down this figure but also verify whether you are getting this particular expression. If you are not getting you are making a mistake, please check your expressions whether you have taken the correct form of T_1, T_2, T_3 and multiply one after the other correctly to get this final expression of T which is what a scaling about an arbitrary point of in space and that is also an example of a composite transformation. We talk about reflection through an arbitrary line, why arbitrary line?

(Refer Slide Time: 00:29:38)



Reflection through an arbitrary line

Steps:

- Translate line to the origin
- Rotation about the origin
- Reflection matrix
- Reverse the rotation
- Translate line back

$$T_{GenRfl} = T_r \, R \, T_{rfl} \, R^T \, T_r^{-1}$$

Well, we have reflection matrices which were reflecting about certain specific axis like you can reflect about x axis, about the y axis or a complete diagonal line which is the equation of the line y equal to x or y equal to minus x as you can think of. Now you have an arbitrary line y equal to mx plus c and I would like to reflect it about that and I hope you remember reflection. We are talking about reflection with respect to a plane or a line and you could have visualized with a mirror about what you mean by reflections in 2D space as well.

Now in this composite transformation you need five different stages, why? Of course we were talking about before rotation and scaling to translate to the origin and back and that must be there, number one. In addition to that you may need to rotate that line or the coordinate such that the arbitrary line coincides with the x axis or the y axis or y equal to x axis.

You need all that because, as like the previous case of rotation and scaling you had the original transformation matrices with respect to the origin. In this case for reflection you have the form of the reflection matrix helps you to reflect an object about certain specific lines y equals x, y equals 0, x equal to 0 that means horizontal vertical axis or whatever it is. So you must make the arbitrary line coincide with some of these special lines and then only you can apply your reflection matrix. Then of course you rotate, translate back as you have done before in the case of rotation and scaling to get back the composite transformation matrix.

So look back into the board, there are five different stages, translate the line to the origin if the line is not passing through the origin, if it is passing through the origin the first and the last step may not be necessary. Otherwise generally translate the line to the origin, rotate the line about the origin, use the reflection matrix to reflect the object or the point reverse the rotation then reverse the translation process back as well.

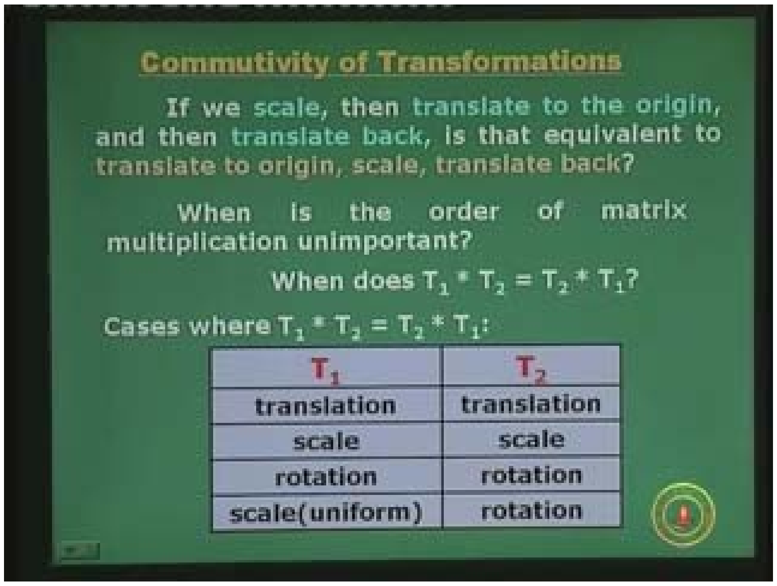
I will talk of general reflection matrix, general reflection matrix, composite matrix, transformation matrix for generalized reflection. I will talk of a composite matrix of a generalized reflection, we talk of translation, then a rotation, then a reflection, then a reverse of rotation which you know because since rotation matrix is orthogonal also this R inverse is nothing but its transpose and of course you also take inverse of the translation matrix at the end. I leave it as an exercise to compute the overall composite matrix in that case.

Now we discuss about commutivity of transformations. That means if you scale, then translate to origin, then translate back some sequence of transformations say arbitrarily chosen, is that equivalent to translate to origin that means can this process be interchanged in the sense that, can the sequence be disturbed and can we have the same result?

If you scale, translate to origin and then translate back is that equivalent to translate to origins? Scale and translate back? Is the order important? That is the question, is the order of transformation which is dictated by the order of matrix multiplication or vise versa. The matrix multiplication dictates the order of transformations or the order is dictated by the sequence of matrix multiplication. Is the order important or unimportant? What do you think?

Well I am talking about scenarios that when does T_1 multiplied by T_2 equals T_2 multiplied by T_1. If you go back to matrix theory or matrix multiplication the order is important. You cannot change the order and get the same results which means in general you will not have, if you have arbitrary matrices and multiply T_1 and then with respect to T_2, T_1 post multiplied by T_2 is not equivalent to T_1 premultiplied by T_2. So that is what is given in this. Is it the same? It is generally not.

(Refer Slide Time: 00:32:20)



Commutivity of Transformations

If we scale, then translate to the origin, and then translate back, is that equivalent to translate to origin, scale, translate back?

When is the order of matrix multiplication unimportant?

When does $T_1 * T_2 = T_2 * T_1$?

Cases where $T_1 * T_2 = T_2 * T_1$:

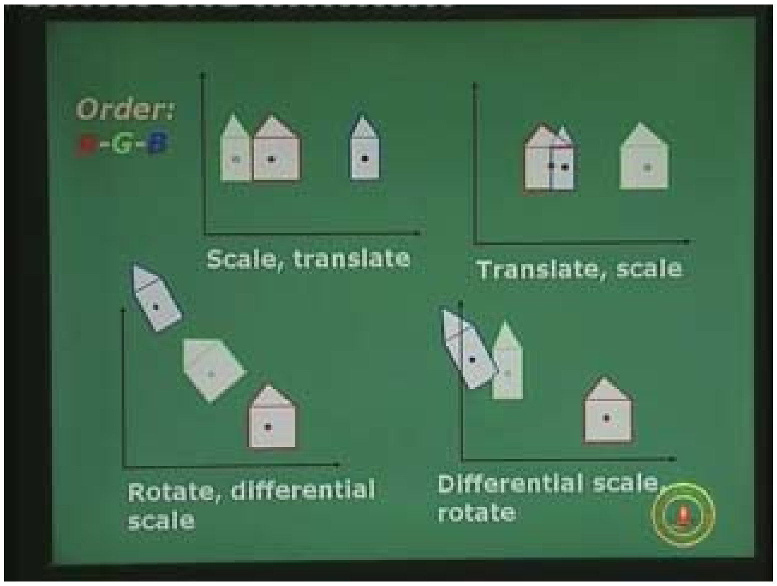
T_1	T_2
translation	translation
scale	scale
rotation	rotation
scale(uniform)	rotation

But in certain specific cases you may have T_1 multiplied by T_2 equals T_2 multiplied by T_1 that means pre or post multiplication of matrices does not make a difference but in certain special cases and in general not. So it is mathematics of matrix multiplication which is dictating that the order of the multiplication as well as the order of transformation is very very important. We cannot jumble up the order and simply say that I am expecting the same result and I should get the same result, in general you will not get it. But in some cases yes, the cases where T_1 and T_2 being post multiplied or premultiplied gives you the same.

We will see in the slide now in this particular table the cases were T_1 multiplied by T_2 is equal to T_2 multiplied by T_1 that means you can take in any order pre or post multiplied. If you take two translations, you take two successive scaling operations, two successive rotation operations or a special case of a uniform scale followed by a rotation. These are the pairs of combinations of T_1 T_2. You check for yourself that the order is unimportant. You can apply it in any order, you can apply the translation T_2 and then apply translation T_1 or scale T_2 then scale T_1 at the same as the scale T_1 and then scale T_2.

I am talking of any T. You should take a pair of transformations in any horizontal row not along the column. That means you can take T_1 as rotation then T_2 as rotation and reverse it you get the same result. I am not asking you to combine translation and scale, take two translations, take two scales, reverse the order you get the same result. That means in these specific cases of four cases as given here, these are four examples only where the order is unimportant. Otherwise in general the order is very very important where you can not swap or jumble up the order and reverse it and expect the same result.

(Refer Slide Time: 00:34:57)



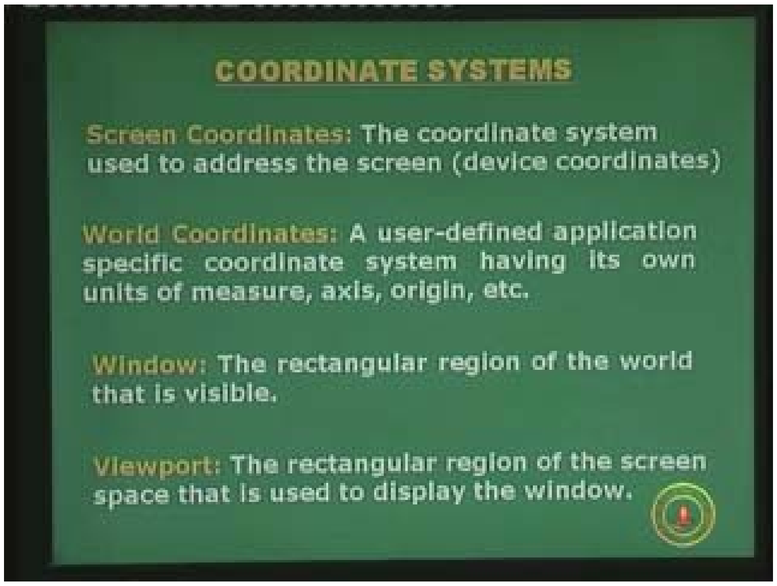
Just to highlight the importance of this order I have taken an example, this example shows the order in which I am scaling and then translating that means you look at the colors the original object is marked as red it is the same house object and then I give it a scaling which gives you a greenish house and then I give it a translation and it gives a blue house. So the red house gets scaled first to the green and then translated to the blue house and if I reverse the order, what I am doing, I first translate from red I get the green and then I scale it down I get the blue house as you can see the final result is different.

Remember, the order is red, green and blue. So you start from red end in blue, reverse the order. As you can say that the final result is different, the final result is marked in blue. This is the final result of scale and then translate this is the final result when you translate and scale.

We will take an other example; rotate and differential scale that means you are first rotating the house marked in red to a greenish position, a position marked by the green house and then give it a differential scale. We are talking of non-uniform scaling that is meant by differencing scale which is the other term that is used and that is the final position given by the house marked in blue. But if you alter it, that means give a differential scale first and then rotation, you can see now that the red house given differential scale or a non-uniform scale gives rise to the green house first, the house marked in green is what I mean and then you give it the equivalent amount of rotation and you land up with a blue.

Again you see that the final result is different. These two examples show that the order in general is very very important. You can not just jumble up the pre or post multiplication operators and expect the same result.

(Refer Slide Time: 36:16)



COORDINATE SYSTEMS

Screen Coordinates: The coordinate system used to address the screen (device coordinates)

World Coordinates: A user-defined application specific coordinate system having its own units of measure, axis, origin, etc.

Window: The rectangular region of the world that is visible.

Viewport: The rectangular region of the screen space that is used to display the window.

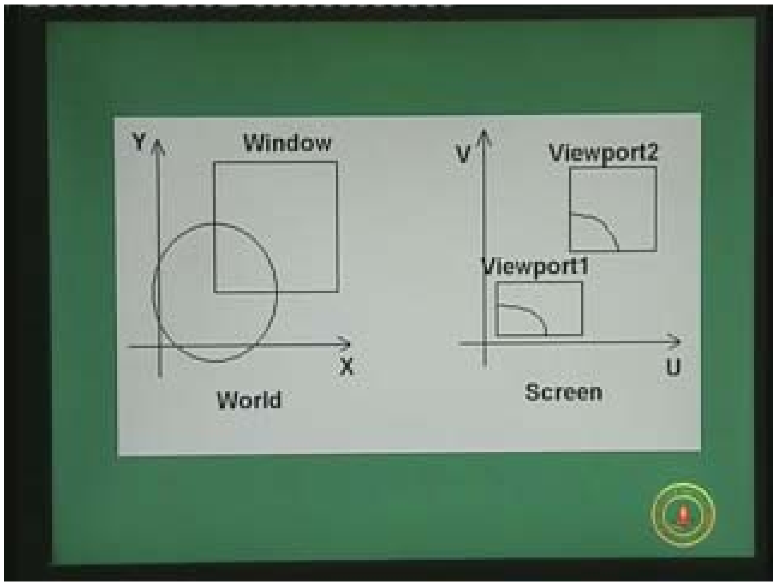
And we talk about coordinate systems, many types of coordinate systems in 2D exists; one is called the screen coordinate system, this is the coordinate system used to address the screen or what are called the device coordinate, the pixels actually reside on the screen that is referred to as screen coordinates. We also have world coordinates which are the user of the program defines the user define application, specific coordinate system having its own units of measure, axis, origin, etc. Those units may not be pixel units, it could be inches, millimeters, kilometers, feet or whatever it could be and some sort of length units is necessary and the user and the programmer knows.

And you have a windowed based coordinate system. It is a rectangular region of the world or the screen that is visible and its own units basically again in pixels in most cases.

Of course, you have a viewport on the window which is a rectangular region of the screen space that is used to display the window. So you have a window of the world I should say and a viewport on the screen. To understand the different coordinate systems and their relations you look into the screen in terms of the world on the system and the windowed coordinate system on the left hand side and the screen coordinate system on the viewport coordinate system on the right hand side.

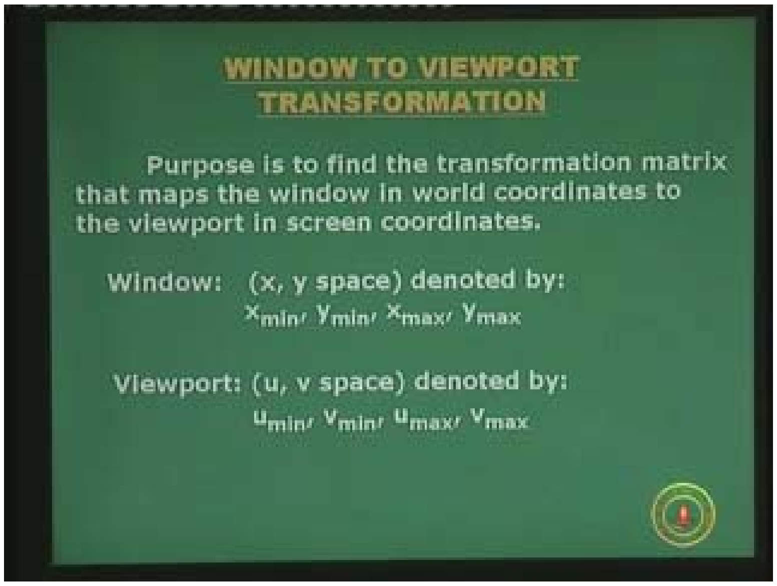
You see that the window is a small part of the world the coordinate system and a viewport is a small part of the screen coordinate system and if you are displaying a small circular path or a circle, the window takes a part of that arc of the circle and you can display that in any of the viewports. The two viewports displaying the windowed window of the world and you can see the corresponding arcs of the circle in two different viewports. You can define that viewport anywhere on the screen.

(Refer Slide Time: 00:38:12)



Now but you need a transformation matrix. So we are still in 2D transformation. You need a transformation matrix to transform from at least the window to the viewport. Of course the window is described with respect to the world and also viewport describe with respect to the screen. So, if this is the world for you, now you could have a small window were you can see only my face, may be see, or you can some characters written on the screen and some part of it or you can assume this also to be your screen. You would like to have a small viewport and you do not want to make anything else visible outside the viewport.

(Refer Slide Time: 40:14)



WINDOW TO VIEWPORT TRANSFORMATION

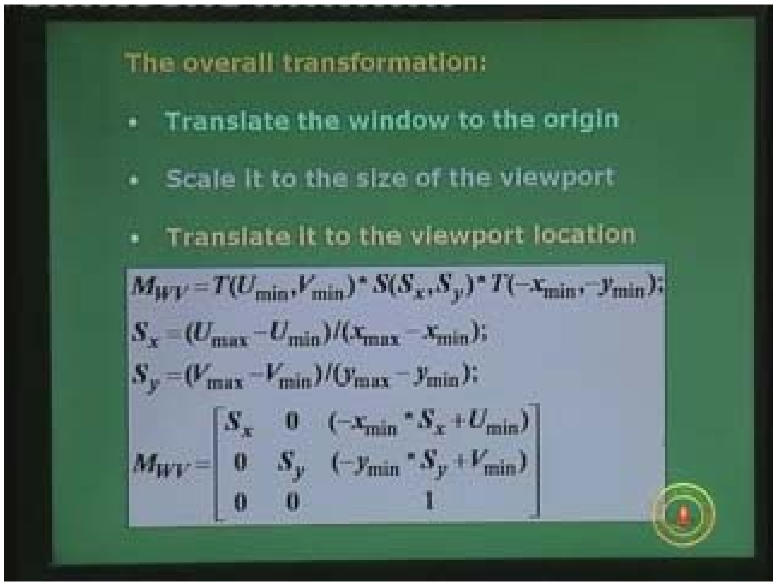
Purpose is to find the transformation matrix that maps the window in world coordinates to the viewport in screen coordinates.

Window: (x, y space) denoted by: x_{min}, y_{min}, x_{max}, y_{max}

Viewport: (u, v space) denoted by: u_{min}, v_{min}, u_{max}, v_{max}

It is something like looking through a window in your house and you see a small part of the world through the window. That is the example of a window or a viewport, the port through which you are able to see and that is what is being talked about and into transformation to transform window to viewport. So the purpose is to find the transformation matrix that maps the window in world coordinates to the viewport in screen coordinate system. And we talk about window x, y spaces system denoted by x min y min x max y max the corresponding minimum and maximum coordinates in x and y direction respectively. And similarly you also have a viewport instead of x, y you typically use another coordinates u, v. Sometimes x, y and u, v are used interchangeably within the device coordinates or window viewport. We will get used to this u, v. And similarly in the u, v space you have a minimum or maximum coordinates defined as u min u max or v min v max respectively.

(Refer Slide Time: 00:39:44)



The overall transformation:

- Translate the window to the origin
- Scale it to the size of the viewport
- Translate it to the viewport location

$$M_{WV} = T(U_{\min}, V_{\min}) * S(S_x, S_y) * T(-x_{\min}, -y_{\min});$$
$$S_x = (U_{\max} - U_{\min})/(x_{\max} - x_{\min});$$
$$S_y = (V_{\max} - V_{\min})/(y_{\max} - y_{\min});$$
$$M_{WV} = \begin{bmatrix} S_x & 0 & (-x_{\min} * S_x + U_{\min}) \\ 0 & S_y & (-y_{\min} * S_y + V_{\min}) \\ 0 & 0 & 1 \end{bmatrix}$$

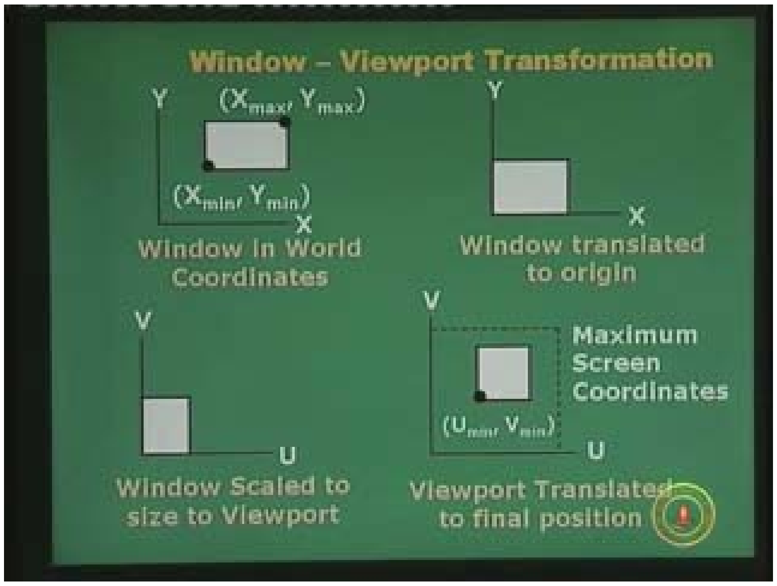
So the overall transformation is to translate the window to the origin and you have to scale it to the size of the viewport and then translate it to the viewport location. These are the three transformations involved in the overall transformation matrix where you want the window to be transformed to the viewport, because you want to display a section of the world in a particular section of the screen which is the viewport.

In general, you want the entire world, world means the entire set of pictures which you are drawing on the screen or for which you have a model basically. You do not want to draw all those pictures on the screen but you want a certain section of your world which you have defined yourself. The world in this case talks of the pictures and objects which you want to draw or model basically using mathematical tools or mathematical functions and you want a part of that to be displayed not may be on the entire screen but to a part of the screen. So that part of the screen is the viewport. The part of that world which you want to display is the window. You want to take this window and put it in the viewport. That is the transformation we are talking about.

And let us look at this transformation matrix as given in this slide. Now we are talking of the overall transformation matrix M_{wv} that is the notation which I have used. If you look back into the slide it consists of three transformations; a translation, scale and translation again. We are already used to these types of transformations. So T with U min V min multiplying by a scaling matrix and then a transformation matrix again and where the scale factor for the scaling matrix is given by the ratio of the maximum minimum coordinates along x and U respectively and along V and y respectively. And if you plug this S_x and S_y into the scale matrix and then multiply T S and T the final form is given at the bottom of your screen.

M is the overall matrix for transformation. The wv subscript indicates window to viewport. So we are talking about overall transformation matrix to transform from window to viewport, M window to viewport is given by this, S_x S_y at the corresponding scale factors for the scaling matrix x_{min} y_{min} U_{min} V_{min} are also known and U_{max} V_{max} is also taken care of by the S_x S_y. This is an example of a transformation in the world to device coordinates. So after we had seen the nature of the transformation matrix which could be used to provide the transformation from the world coordinate systems or the programmers coordinate system or the users coordinate system to a viewport within the device or screen coordinates. We will take an illustration now of an object which will do this task. As we go along we will take an illustrative example of what do you mean by this transformation.

(Refer Slide Time: 00:43:15)



Window – Viewport Transformation

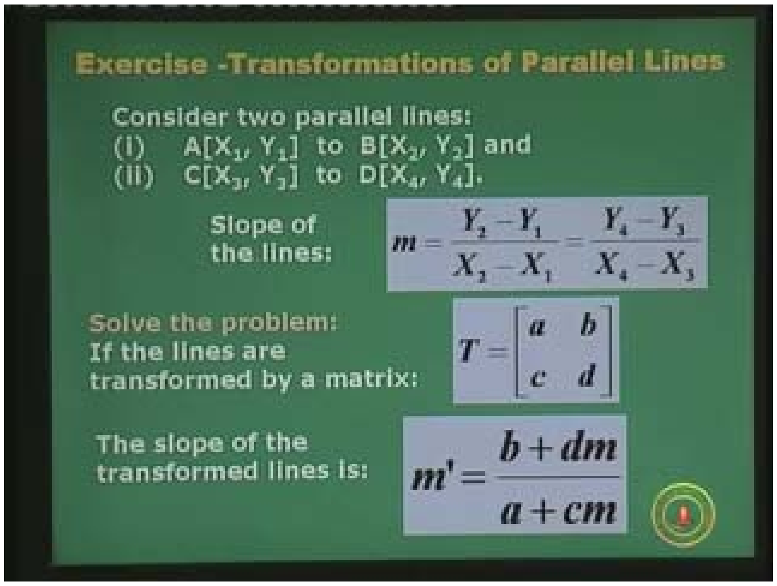
Y
(X_{max}, Y_{max})
(X_{min}, Y_{min})
X
Window in World Coordinates
Y
X
Window translated to origin
V
U
Window Scaled to size to Viewport
V
Maximum Screen Coordinates
(U_{min}, V_{min})
U
Viewport Translated to final position

So window to viewport transformation we had seen, the matrix representation is a combination of translation, a scale and a translation back again and we take a an object here, a simple square set object is taken as you see in the slide here which is a small square shaped object which represents the window in world coordinates. It typically means a programmer or a user may have all his pictures and objects drawn all over in this world, over all X and Y cos values, it could be positive and negative. But at the end what he would have wanted is or he would have liked or desired is, now at any given point of time let us say he wants his objects drawn from X_{min} Y_{min} to X_{max} Y_{max}, all the objects drawn here to be put in a particular position in the viewport, somewhere in the viewport, it must be put in the viewport. The viewport belongs to a small part of this screen. And the first action is to translate it to the origin that is X_{min} Y_{min} becomes 0 and X_{max} Y_{max} which is the top right corner of the box or the window will now become X_{max} minus X_{min} Y_{max} minus Y_{min} which is the top left corner coordinates after translation. You want to scale it back because the size of the viewport may not actually match with the size or shape of the window. The size is more important rather than the shape. It typically takes a rectangular shape so size is an important factor.

Hence scaling matrix will come into play and it will implement the scaling. It will scale the translator window to the origin to another shape or size at the origin itself and then we need to translate it back to the point were the viewport in the screen coordinates. We are already into U and V after the scaling and we now translate in the U V space where the point U_{min} V_{min} is given by the left bottom most coordinate of the viewport. That is the viewport bottom most coordinate in the U V space where we wanted the viewport. This is an example of the window to viewport transformation as it happens using the transformation matrix, the composite transformation matrix.

We have seen lots of example of composite transformation matrices. Now we will take a small exercise about transformations of at least one case or more than one line, say let us take transformation of parallel lines. I will probably encourage you to take transformation of intersecting lines also. But let us say you have two parallel lines and of course anywhere it is parallel. Let us say there are two parallel lines and I want to give it a transformation that means I want to rotate both of these lines simultaneously, I want to translate them, I want to scale them, do some transformations. We are talking of a general transformation matrix.

(Refer Slide Time: 00:46:24)



Exercise -Transformations of Parallel Lines

Consider two parallel lines:
(i) A[X_1, Y_1] to B[X_2, Y_2] and
(ii) C[X_3, Y_3] to D[X_4, Y_4].

Slope of the lines: $m = \frac{Y_2 - Y_1}{X_2 - X_1} = \frac{Y_4 - Y_3}{X_4 - X_3}$

Solve the problem:
If the lines are transformed by a matrix: $T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

The slope of the transformed lines is: $m' = \frac{b + dm}{a + cm}$

Let us consider two parallel lines as given in the screen; one runs from A to B with coordinates X_1, Y_1 to X_2, Y_2 as given in the screen here and the second line runs from coordinates C to D X_3, Y_3 to a point X_4, Y_4. That means there are two lines A B and C D. And I want to transform these lines by a general transformation matrix A B C D.

Now of course I have not taken homogeneous coordinates here, it does not matter; you can work with the general transformation matrices in 2D without the translation let us say or translation also we know what to do. There is actually no problem with translation when you are translating parallel lines. They will remain parallel anyway. But what about others; if you scale, if you rotate, if you reflect, if you give it a shear, so in transformation of parallel lines if we do any sort of transformation what happens?

I will not solve the problem entirely for you but will give you hints and direct you to solve it. The slope of the lines can be computed by this expression; m is the slope of the line, when given two end points of a line you can always compute the slope. I hope you know this from fundamental mathematics or geometry is given by Y_2 minus Y_1 divided by X_2 minus X_1 or Y_4 minus Y_3, X_4 minus X_3 so both are the same, that is the slope of both the lines A B or C D. You solve the problem by transforming these lines by a transformation matrix T given by A B C D.

So how you solve it? What I would request you to do is you transform a by T, b by T, c by T also and also d. All the four coordinates, that means the two pair of end points or the corresponding pair for the each line transform it by T and then compute the new slope. Compute the new slope I would say that is m prime from the transformed m points and try to compute the slope of the transformed lines m prime as a function of these elements a b c d and m. That is the original slope of the line. And you will find that the slope of the transformed lines, both the lines get an equivalent transformation. Their slope also changes in a same way that means from m you reach another m prime given by this expression.

This is the exercise which I am leaving to you at the end of this talk, almost that the m prime can be written as an expression given by b plus dm, a plus cm respectively. As you see, if you look back into this expression here, the functional form for m prime, it depends on the original value of m that is the slope of the parallel lines before transformation and the elements of the transformation matrix. It does not depend upon the coordinates. That means you can take any points a and b on the line, any point see on those two parallel lines, the transformed lines, the two parallel lines which had a slope m after transformation will also have the same slope m prime. That means parallel lines after rigid body transformation will remain parallel. They will not change.

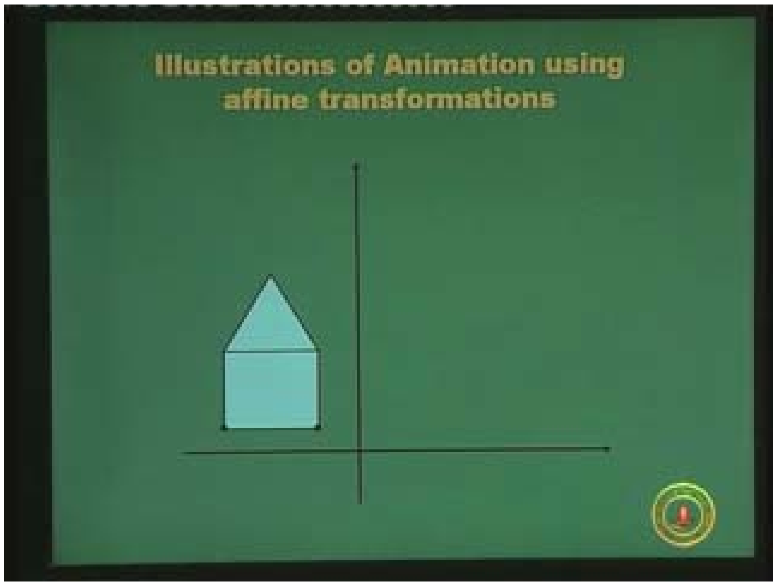
Rigid body transformation has this property. You take two parallel lines of an object or any object shape, it could be a house or any other structure, give it a rotation, give it a shear, give it a reflection, or even translation for that matter and the two parallel lines remain parallel. The slope of that line changes, that is given by the expression here which you must workout. Solve it yourself and see if you cannot solve it you can contact me by an email or send it by post for me to correct it or you can contact your teachers but what the lesson we learn is parallel lines transformed remains parallel but their slope changes that means the they had a slope m and it is now become m prime which is different slope but again remain parallel. This is interesting.

I would also ask you to take another exercise to find out if you have two intersecting lines and their intersection point is computed and after transformation you get a new point of intersection and the entire object which is transformed by an equivalent transformation matrix, the corresponding intersection point is also transformed by the same T. This is derived from the law or the concept that all points of rigid object are transformed equivalently by the same transformation matrix which is the transformation matrix for the entire object, or for all points, or for all lines, or for all parts of the object. As long as the body is rigid it does not change its shape or size. We did talk about deformable structures which we are not discussing here.

We are talking rigid structures which we see in front of us. Not like the cloth or the paper which can be deformed. We are talking of rigid objects and when they are transformed rigidity is maintained, internal structural details are maintained and by transformation it takes the points or the object to some other point. But their internal structural details are at adjacency and their distances between any two points, the corresponding slope relations of two lines, any parts of that object remains the same. That is the lesson we learnt from rigid object transformation.

We will wind up with simple examples of transformation once again just to enlighten you if you have done with a lot of the mathematics you must trust me, you have to believe me that this is the beginning of the mathematics of this course. As we go along with more and more algorithms, we will have more and more of this mathematics, more to do with linear algebra in some cases, algorithms, differential and mostly vector calculus and vector geometry and concepts like that as we move along. And we will go ahead with some simple nice illustrations, very simple but in the sense that the examples illustrate the effect of transformation which is fundamental to animation.

(Refer Slide Time: 00:50:05)

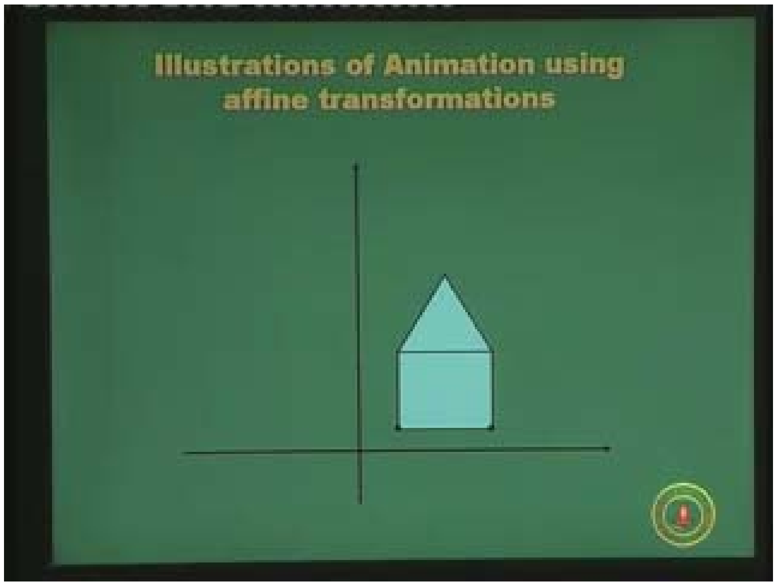


Illustrations of Animation using affine transformations

When you are talking about animation which we will discuss at the end of this course, animation involves motion or some sort of movement and movement along a path. So you must define the path, define the pattern of movement and it is that you need the transformation matrices to in fact move any parts of the object or the object by itself. We will see how transformations are used to create animation with the help of simple structures.

Let us take an illustration here, I take the same house structure and I wanted to rotate it about the origin. It will create an animation type of motion but you have to assume here that I am doing it incrementally. I am doing the amount of rotation in a step by step manner using small thetas and if I do it very fast this is the effect I will get. So it rotates by not a smooth motion, I was trying to create an effect of an animation. I go back and give you the motion again in the example of rotation or a sequence of a small rotation or steps giving it a rotation. So in effect it has rotated and it has created animation.

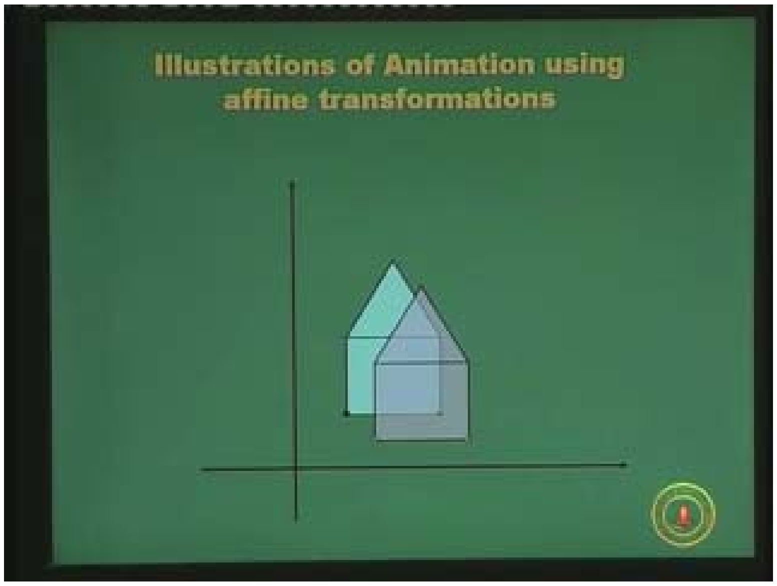
(Refer Slide Time: 00:51:08)



Illustrations of Animation using affine transformations

We will see another effect here where the object is scaled and rotated simultaneously, rotated about its centroid or about some point and within the object also and scaled. It was small object and it is enlarging out. I will repeat the process again, simultaneously it is enlarged, zoomed out and as well as rotated.

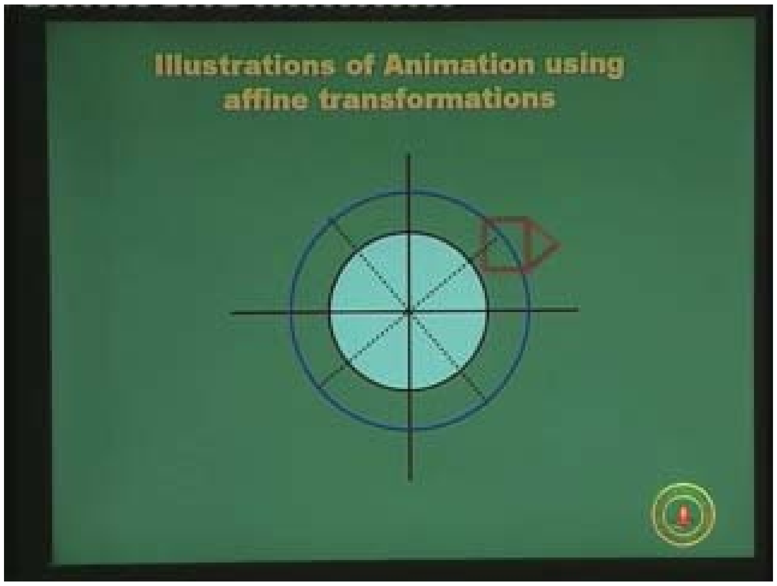
(Refer Slide Time: 00:51:27)



Illustrations of Animation using affine transformations

These are the two objects; one coming through a spiral path another coming through a scale and scalar zooming effect along with rotation which we have seen earlier. I play it again, two objects simultaneously joining but following two different paths of transformations.

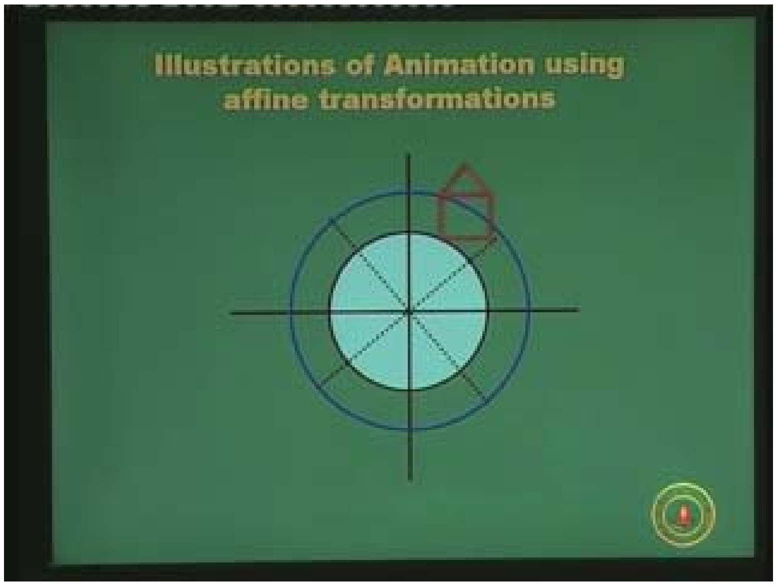
(Refer Slide Time: 00:51:46)



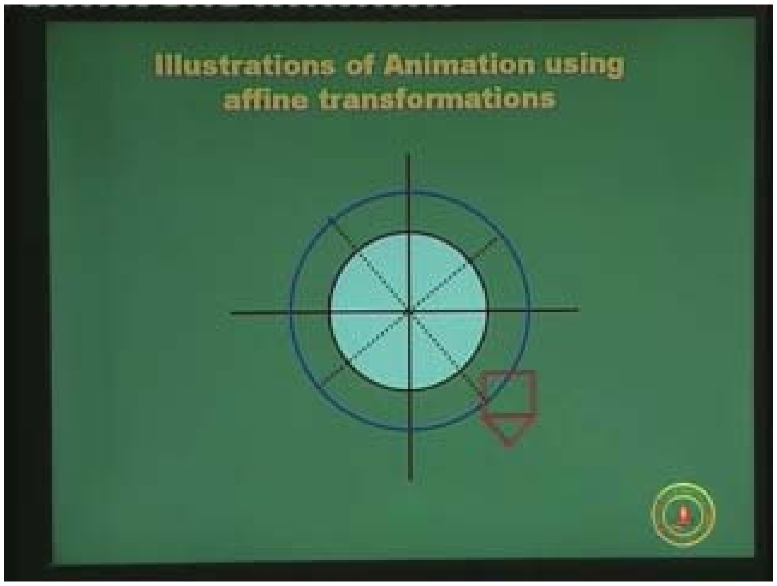
Illustrations of Animation using affine transformations

And the very last one is the simple case of rotation. I think I showed the structure earlier. You see a structure of a house which rotates about the origin and then of course also after rotating about to 90 degrees it also changes its position at that time. That means it rotates about itself.

(Refer Slide Time: 00:51:49)


Illustrations of Animation using affine transformations

(Refer Slide Time: 00:51:51)


Illustrations of Animation using affine transformations

I will repeat that process once again; rotation by origin rotates about itself rotates about the origin at 90 degrees, rotate itself, rotates about origin, rotate itself, rotates about origin by 90 degrees. These are the sequence of examples of transformations, rotations, we have seen scaling and or a combination of rotations, scaling and then of course if you have multiple objects as in the previous example, multiple object paths can be created and you can have multiple objects animation.

So we stop the lecture on two dimensional transformations right now here. In the next lecture we will start with three dimensional transformations where we discuss objects which will be transformed in 3D and interestingly we will find out along with transformations in 3D which involves all the transformations like translation, rotation, scale, shear and reflection. You have a special case of transformation in 3D which does not exist in 2D. You can almost guess it now, it is what we call as projective transformations, perspective orthographic, asymmetric and various types of projective transformation exist only in 3D. There is no scope of that or no application of that in two dimensional transformations. In two dimensional transformations we have five different categories such as translation, rotation, scale, shear, reflection and in 3D we have this entire five plus projective transformation. We will see when we go to the next lecture.

Please solve these examples of 2D transformation problems which I have given here inherent within the lectures itself. Thank you very much.