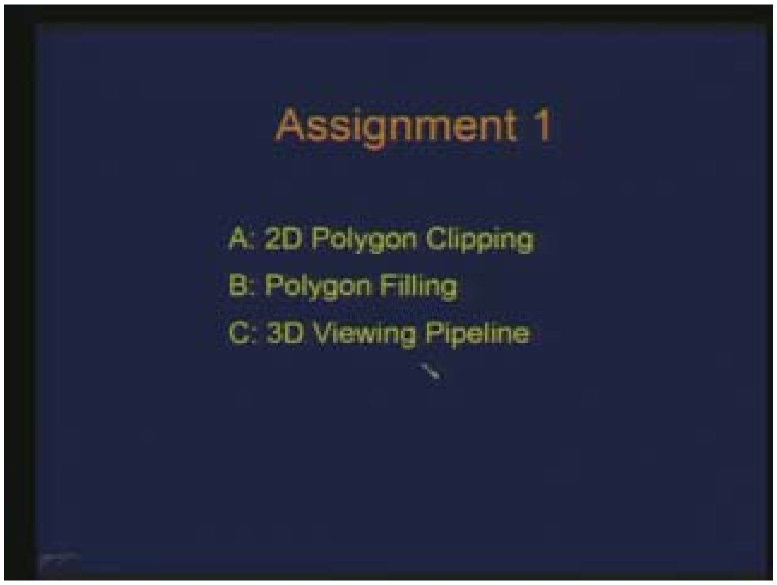
Introduction to Computer Graphics
Dr. Prem Kalra
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture # 11
Assignment - I

Today we will just go through the various components of the assignment step by step. The assignment actually has three components primarily.

(Refer Slide Time: 02:14)



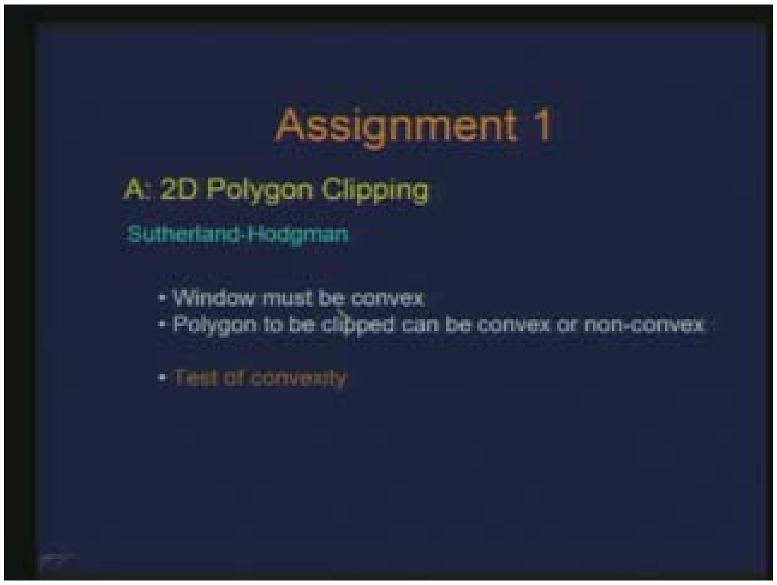
Assignment 1

A: 2D Polygon Clipping
B: Polygon Filling
C: 3D Viewing Pipeline

The first component deals with the polygon clipping namely the 2D polygon clipping. The second component is about filling of the polygon. These two components need to be linked. Once you have done the clipping you should be able to do the filling of that clip polygon. And the third component is the entire viewing pipeline. So these are the various viewing transformations we have studied. So let us look at the input output specification for these components. The first one is the polygon clipping.

Basically you are required to do the implementation of Sutherland Hodgman polygon clipping which works for an arbitrary window which is of a convex type so the window would be convex whereas the polygon which would be clipped can be convex or non convex so you have to show both the cases.

(Refer Slide Time: 03:03)



Assignment 1

A: 2D Polygon Clipping

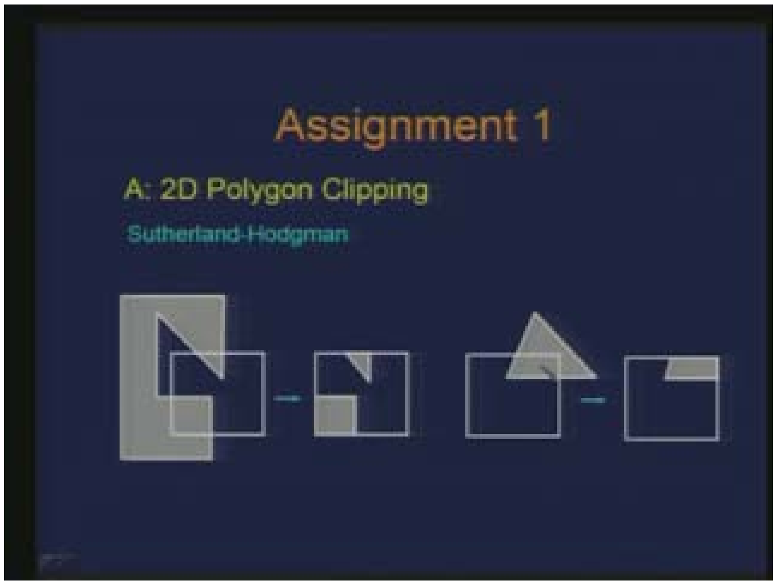
Sutherland-Hodgman

- Window must be convex
- Polygon to be clipped can be convex or non-convex

- Test of convexity

And while you are doing this test of clipping you need to find out whether the particular polygon which has been given to you is convex or not convex. Therefore there is a test of convexity which you need to perform. For non convex windows obviously the standard Sutherland Hodgman would not work. So the test of convexity is the specification or the requirement for the assignment. We have seen how to do this. Basically what we are expecting is that we have seen earlier we have some polygon which is non convex here or convex here and the window is a convex window so here we have considered a rectangular window and this is the result which is obtained.

(Refer Slide Time: 04:28)



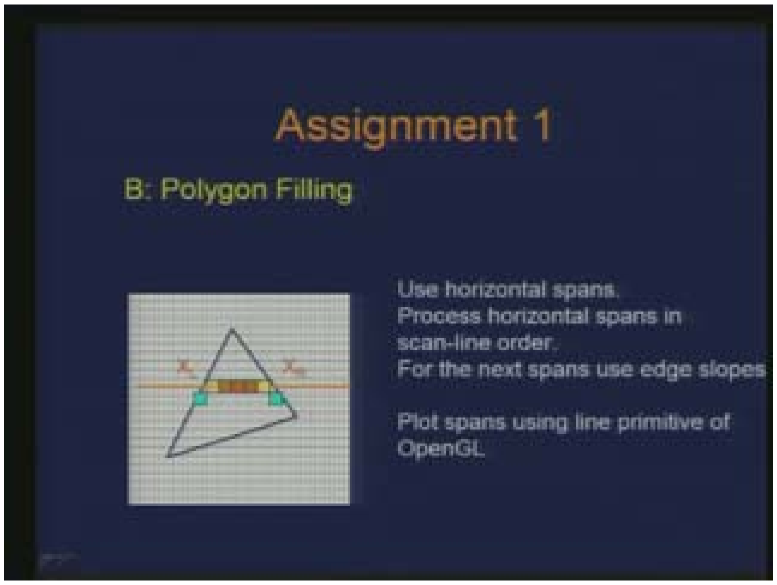
Assignment 1

A: 2D Polygon Clipping

Sutherland-Hodgman

So, for this part of the assignment I have filled this polygon but basically dealing with the lines or the edges of the polygon. When you specify the window I have the notion of these edges, these are the input given by the user. You can specify an order of the specification of the vertices which would in turn determine which is inside or outside. If I give in one order anticlockwise or clockwise that defines what is inside and what is outside. So this is the input specification you give and have that as a convention to be able to decide which is inside and which is outside. So this is the kind of result we are expecting out of the polygon clipping.

(Refer Slide Time: 06:26)



Assignment 1

B: Polygon Filling

Use horizontal spans.
Process horizontal spans in scan-line order.
For the next spans use edge slopes

Plot spans using line primitive of OpenGL

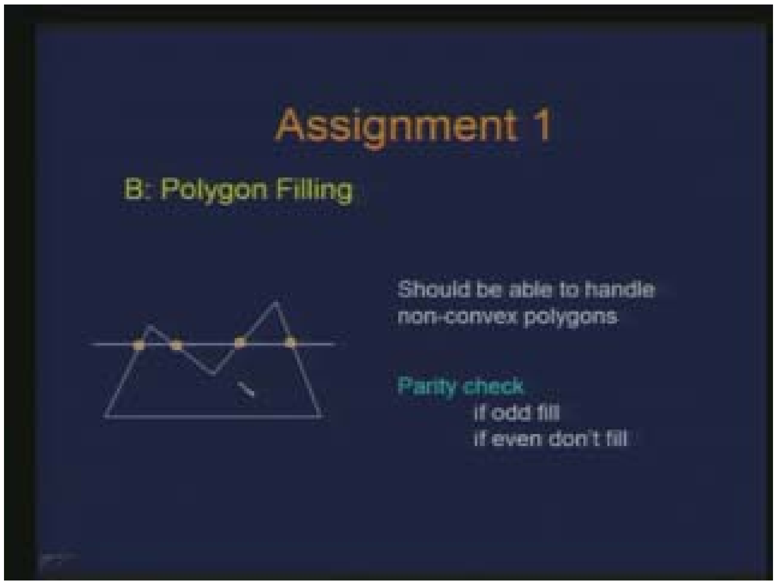
The next component is polygon filling. Again when I say that you are giving the number to the end points of the polygon basically you are specifying the vertex list and the corresponding polygon. And you may actually have a data structure which allows you to have multiple polygons because when you use the description of a scene later on then you can use the same data structure. So try to build things modular so that you can use them later.

Basically we have looked at the way the algorithm works, so there are two algorithms we studied. The first one was the scan line conversion where you go scan line by scan line and find out the intersection of that scan line with the edges of the polygon and then look at the spans of those scan lines which are intercepted within the polygon and fill them, that was the algorithm.

The other algorithm we studied was basically seed filling. So you supply a seed find out the neighborhood of that seed and then propagate the filling from that seed and you can consider either a pixel of a type having four connected neighborhood or you can also consider eight neighborhood pixel. Here you will find out something like the points XL and XR of a span within the polygon and then you will actually render this line using open theory. Here you can use the primitive of OpenGL like a line and just render that.

But you have to account for all the possible ways in which you can optimize the computation. So the scan line coherence for instance is required and that is what you are doing finding out these spans. From one scan line to another scan line when we move it is called the inter scan coherence which also you exploit using the edge slopes. Therefore all those things have to be in your implementation. You should be able to handle both convex polygons and non convex polygons. For instance you should be able to handle this.

(Refer Slide Time: 09:35)



Assignment 1

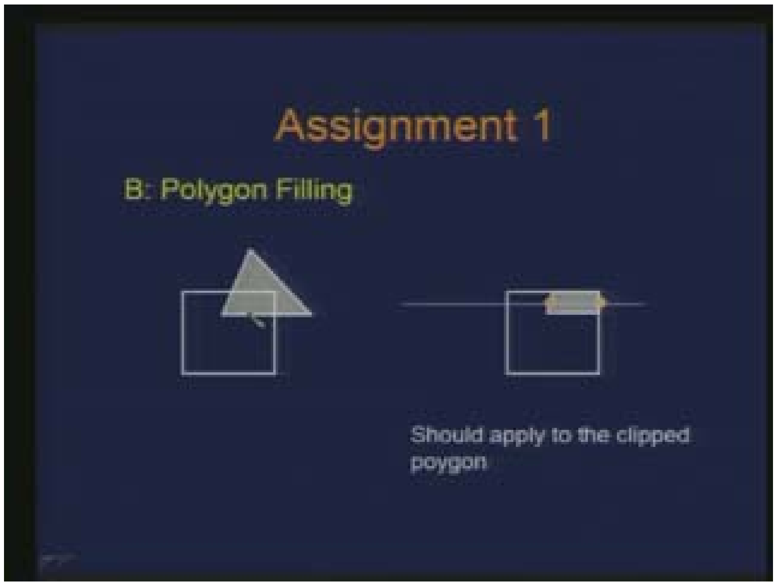
B: Polygon Filling

Should be able to handle non-convex polygons

Parity check
- if odd fill
- if even don't fill

And here also the special case is you need to handle when the scan line passes through a vertex. This particular thing is done through a parity checking. The seed based method is little interesting. So you also need to basically connect these. What we are doing for clipping is the input given for filling.

(Refer Slide Time: 11:10)


Assignment 1

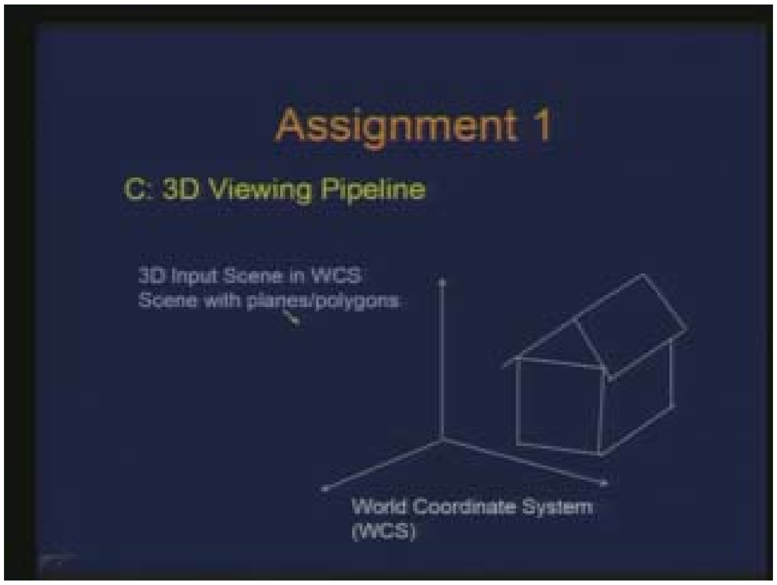
B: Polygon Filling

Should apply to the clipped poygon

Therefore by this way if you perform a clipping on this is the clip polygon and you should be able to apply your filling algorithm on this. Next we move to the third component which is the viewing pipeline. In this viewing pipeline what we are basically trying to achieve is through the interface which we have studied.

Interface means the input specification the way user would give you and then you do the entire pipeline. So the input specifications are very important. Therefore the world or the scene has to be described in world coordinate system something like this. (Refer Slide Time: 12:12) So this is the coordinate system you have and you have a scene description. And the scene could be just the collection of polygons which you can handle at this time. So you can either build that scene yourself or you can import it from somewhere. Import means there might be some scenes available to you in some file format were you have the list of vertices and the polygons constructed using those lists.

(Refer Slide Time: 12:20)



Assignment 1

C: 3D Viewing Pipeline

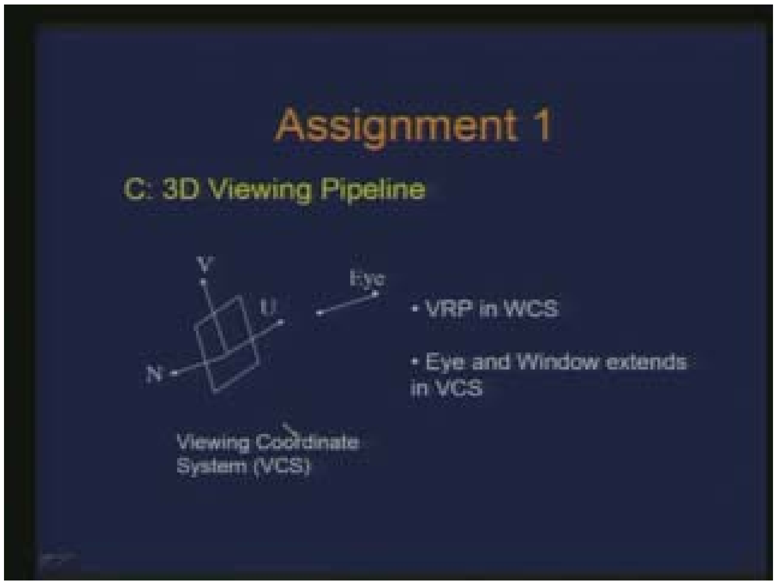
3D Input Scene in WCS
Scene with planes/polygons
World Coordinate System
(WCS)

It is a polyhedral object or polygon mesh. So basically object is the collection of polygons and that may be described through just a list of vertices and the corresponding polygons. So such scenes may be available so you just can import them. Or just through some interface or even through file input you build the scene then you give the end points of the polygon. So here of course you need some data structure to describe your polygon. So this specification of the scene we call it in world.

Later on we will also deal with this hierarchical modeling where you have the models of the objects described in their own coordinate system like a model coordinate system and then you have the scene building using those coordinates so from modeling to the world of the scene coordinate system and then use some sort of a scene graph to describe your scene.

For the purpose of this assignment I have just a flat representation of objects. So the scene is in world coordinate system and then we also have a viewing coordinate system. A viewing coordinate system is basically described by specifying the view reference point given in world coordinate where the origin of the viewing coordinate system is located which is the view reference point which could be considered as the origin of the viewing coordinate system.

(Refer Slide Time: 14:53)



Assignment 1

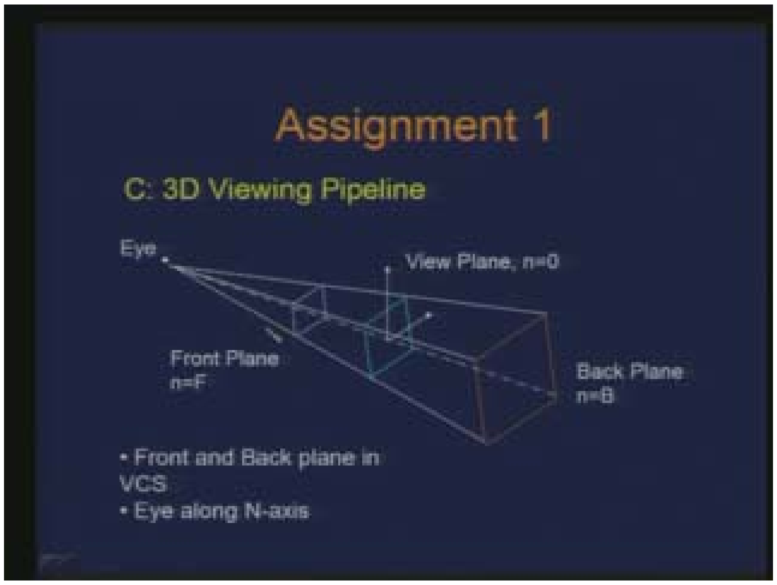
C: 3D Viewing Pipeline

V
U
N
Eye
Viewing Coordinate System (VCS)

- VRP in WCS
- Eye and Window extends in VCS

And then you have these U V and N directions for the viewing coordinate system. For instance, the direction V should be obtained from the up vector defined by the user. So there is a notion of up vector which would be given by this user and that is what we should use in order to obtain the view direction for instance. And the extents of the window are basically defined in viewing coordinate system. So the limits of the window size are in viewing coordinate system. The location of eye the respect to the viewing plane should also be specified in the viewing coordinate system. These are the differences in definitions of a viewing coordinate system as well as the world coordinate system.

(Refer Slide Time: 16:51)



Assignment 1

C: 3D Viewing Pipeline

Eye
View Plane, n=0
Front Plane n=F
Back Plane n=B

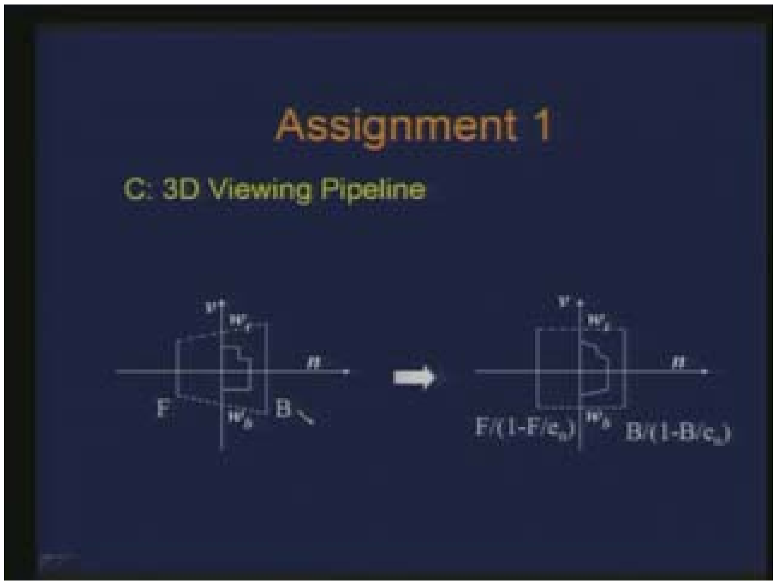
- Front and Back plane in VCS
- Eye along N-axis

So through the proper transformations world to view we eventually get something like this so the viewing frustum gets built. You also need to specify the front or the near plane and the back or the far plane of this view frustum.

Sometimes the view plane itself is considered as the near or the front plane. So there is a possibility for you to consider eye lying along the N axis that is were you will get such a configuration or you may also have eye off the N axis. Besides the transform it will not change much but it will change something when you need to render them. Whenever you have each of these transformations happening as a user from outside I should be able to see what process has changed what, something like a simulation.

What I expect you to have is the simulation of this viewing pipeline. There might be some changes there with respect when you make this eye to be off N axis. Obviously once we have given this view frustum we do the perspective transformation which changes this to this. Basically the center of projection which was lying here goes to infinity now and that is what the perspective transformation does and you have this distortion here.

(Refer Slide Time: 19:44)

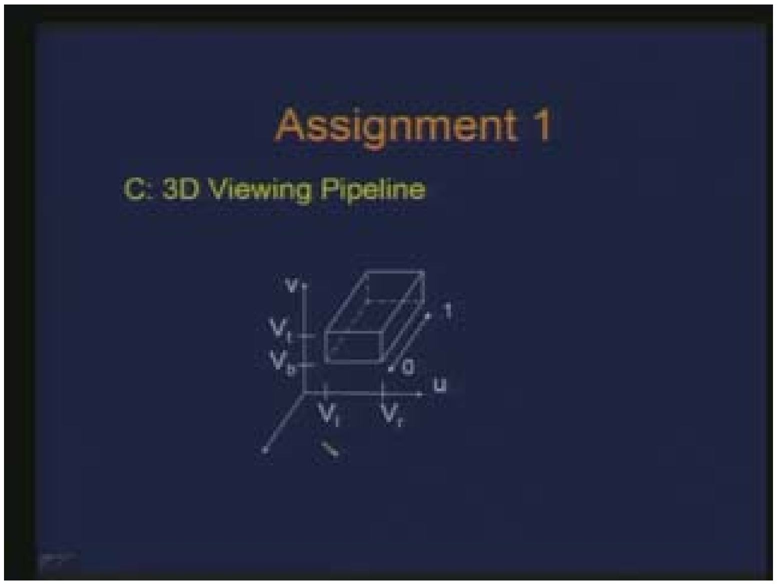


Assignment 1

C: 3D Viewing Pipeline

Then it requires that you need to put this into a normalized coordinate system where you put everything between 0 and 1 for the depth.

(Refer Slide Time: 20:17)

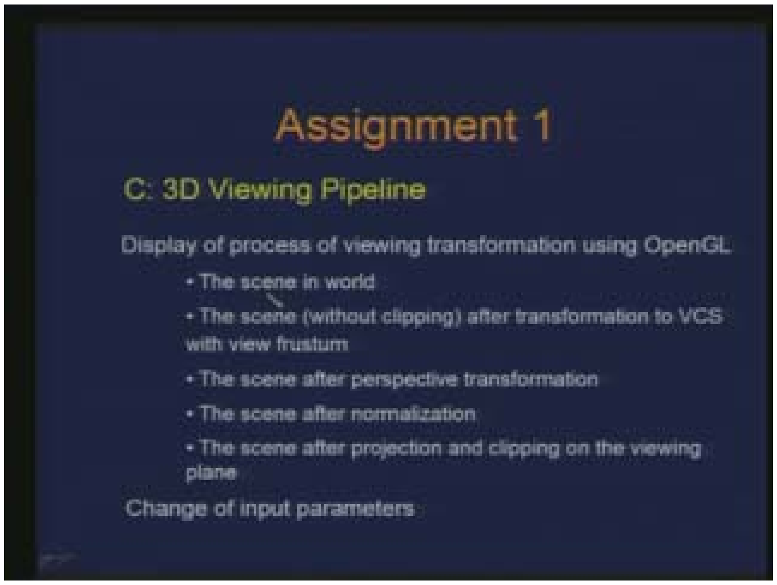


Assignment 1

C: 3D Viewing Pipeline

And these extents which would define through the window are now mapping it to something like V_l V_r V_b V_t and at times this is also considered like a view port of your viewing pipeline. Let us now see what is expected as the output from this. So what we have said is that the scene is defined in worlds so I should be able to see the scene independent of your viewing pipeline. The pipeline which is intended to be implemented I should be able to see the scene just using OpenGL.

(Refer Slide Time: 21:24)



Assignment 1

C: 3D Viewing Pipeline

Display of process of viewing transformation using OpenGL

- The scene in world
- The scene (without clipping) after transformation to VCS with view frustum
- The scene after perspective transformation
- The scene after normalization
- The scene after projection and clipping on the viewing plane

Change of input parameters

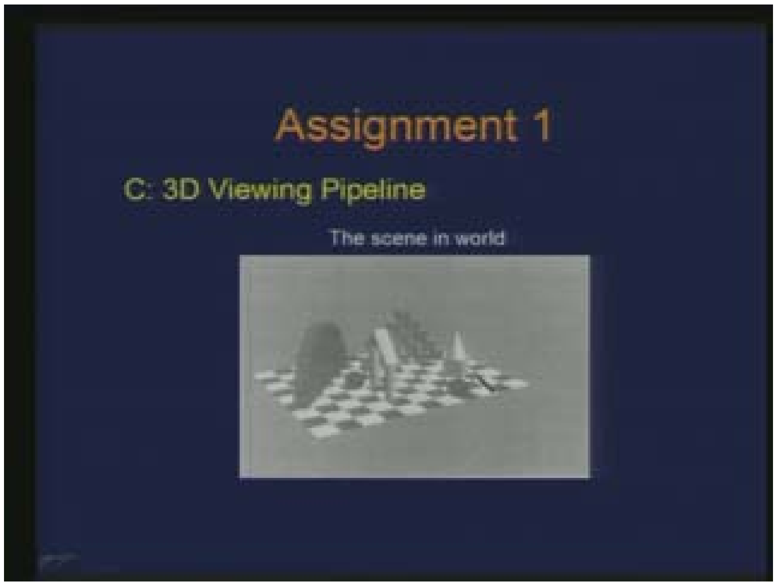
So there is a combination of your viewing pipeline and what OpenGL does. So it is a simultaneous learning, you write your own viewing pipeline and also learn about what

OpenGL does for you. So it is a just simple rendering of the scene using OpenGL and that is what the world is. So I should be able to see the scene.

The second thing is the scene without clipping after transformation to viewing coordinate system with its view frustum. So I should be able to see that from an outside view point. The way the frustum is going to get built I should be able to see it from some other view point using OpenGL. So, that viewing pipeline becomes like a world for the OpenGL.

Here is an example. This is the scene in the world. The scene is defined in this world using this plane, these pillars, something there, something there, these are boxes there etc so this is the scene defined in the world and I am just seeing from one view point choosing OpenGL.

(Refer Slide Time: 23:25)



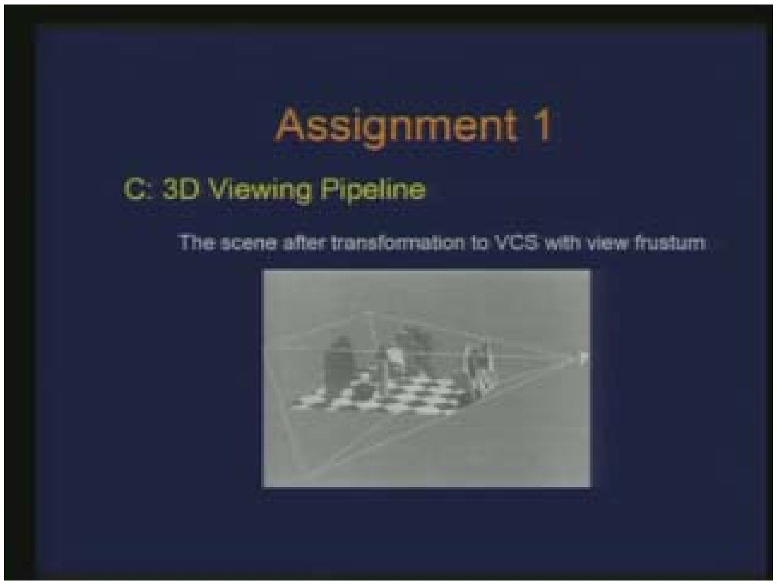
Assignment 1

C: 3D Viewing Pipeline

The scene in world

Now I want to see this also. Now what have you done? Basically you have applied all the viewing coordinate system changes, you have specified this eye and this is the view frustum which gets built up using the transformations we have conducted. So here of course the near plane and the viewing pipeline have been considered to be the same. This is my back plane, this is my front plane and this is the scene with respect to this coordinate system and again I am able to see this.

(Refer Slide Time: 23:49)



Assignment 1

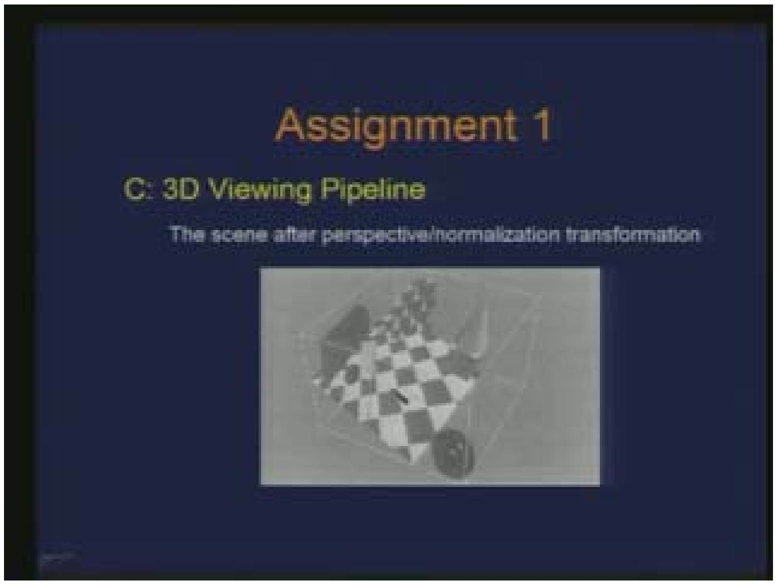
C: 3D Viewing Pipeline

The scene after transformation to VCS with view frustum

Basically now we have seen the scene after transformation to the viewing coordinate system with the view frustum and where the camera is or where the eye is. Then we would like to see the scene after perspective transformation and normalization and then we see the scene after projection and clipping of the viewing plane. That is where you will bring in your clipping because we are basically considering clipping in 2D, you could have done clipping in 3D as well but that is going to be bit hard. Had there been only line clipping then it would have been fairly easy. …………there of course we are not asking for the filling part because there you are talking only the wire frame rendering of the object

And of course there has to be a facility by which you can change some of the input parameters. I should be able to change the location of eye, I should be able to change the up vector, I should be able to change several other parameters which are specified by the user and see the result. After this is what we said that you should be able to demonstrate the perspective transformation and the normalization. So, that has to be done in two stages but the result which I am showing here is actually a combination because you basically take one parallelepiped to another parallelepiped it is not a big difference. But this is what you would have once the perspective transformation and the normalization has been done. So this is how the scene looks in this cuboid.

(Refer Slide Time: 27:12)



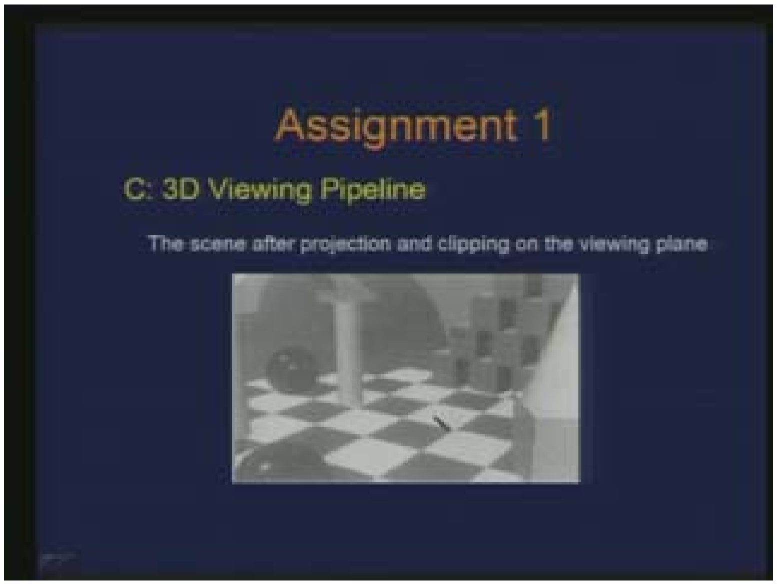
Assignment 1

C: 3D Viewing Pipeline

The scene after perspective/normalization transformation

Thus, you are able to see the cuboid as well as the scene within it. So may be to start with you should consider your scene in such a way that everything comes within this cuboid because if you take a large scene and then the view frustum which is built would cut it so normally there has to be a clipping. This is what we have and the last thing would be the image. That is basically the image on the viewing plane. Here you would perform the necessary clipping and color them. Hence, what are we not considering here is we are completely discarding the hidden surface part. We are not incorporating it so you just take the list of polygons which get intercepted within it and just paint them or render them or color them using the filling. So what it may not have is the right depth or the occlusion which you would otherwise have here.

(Refer Slide Time: 28:40)



Assignment 1

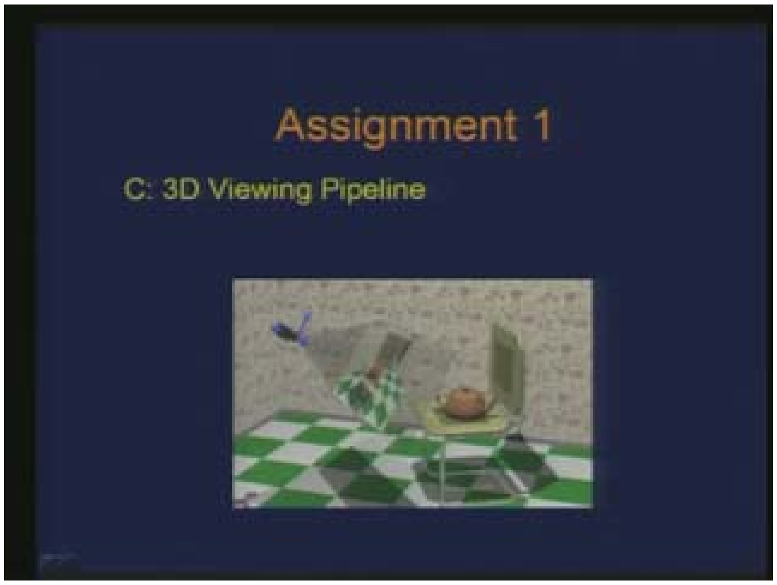
C: 3D Viewing Pipeline

The scene after projection and clipping on the viewing plane

Maybe to start with you can consider a sparse kind of a scene where the overlaps are not there and therefore you see it right. Otherwise what is going to happen is, if you happen to paint the polygons with respect to this object later than this you will see this and not this although this is in front of this. Hence, this ordering is being discarded at this point of time because we have not dealt with hidden surface elimination. Thus, at the end of this whole pipeline what we have is this image. But you have to show the entire process. This is just an illustrative example. Right now just consider planner objects just polygons.

So the scene complexity is of no criteria for this assignment. The functioning with respect to the various stages of the pipeline are important and not the complexity of the scene. You can actually consider the kind of scene I showed in the beginning. Just take a couple of boxes, because a scene complexity is not something you should worry about at this stage. But theoretically speaking it does not matter if you can handle ten polygons right you can handle hundred polygons or thousand polygons right and it should not affect. Here is another example, here this is what we are expecting and this is what we want to do in a way.

(Refer Slide Time: 32:30)



Assignment 1

C: 3D Viewing Pipeline

This whole thing is now given within the view within that scene so the camera is specified here, the viewing plane is specified here and that is the part you see of this scene and this is being done using the transparency and so on. The idea is that you are able to demonstrate the whole scene to anyone and simulate the process.