Computer Graphics

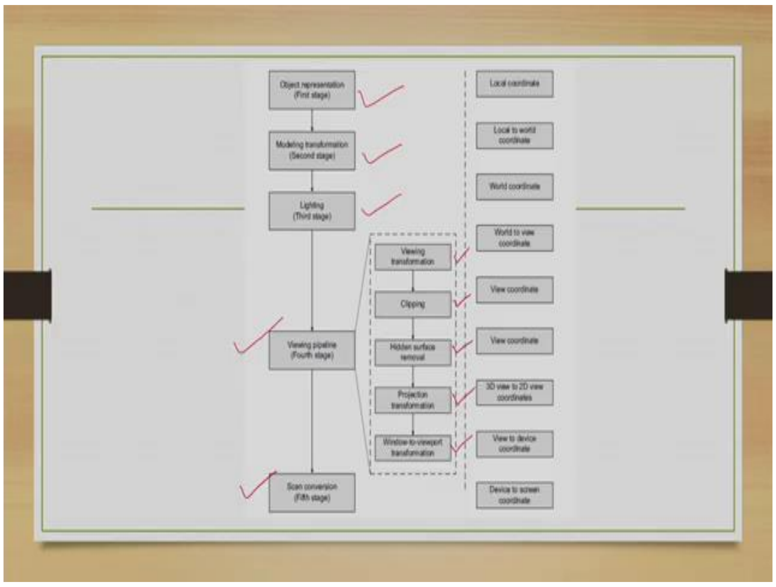
Professor Dr Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture 9

Space Representation Methods

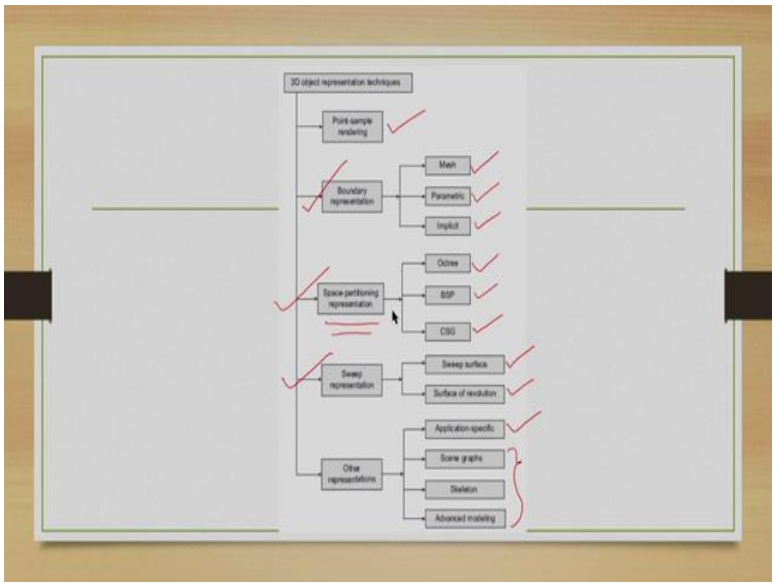
Hello and welcome to lecture number 9 in the course Computer Graphics. We are discussing the stages of the graphics pipeline. As you may recollect, there are 5 broad stages, the very first stage is the object representation, and we are currently discussing on the object representation technique.

(Refer Slide Time: 0:55)


Object representation (First stage)
Modeling transformation (Second stage)
Lighting (Third stage)
Viewing pipeline (Fourth stage)
Scan conversion (Fifth stage)
Viewing transformation
Clipping
Hidden surface removal
Projection transformation
Window-to-viewport transformation
Local coordinate
Local to world coordinate
World coordinate
World to view coordinate
View coordinate
View coordinate
3D view to 2D view coordinates
View to device coordinate
Device to screen coordinate

This figure may help you recollect the stages. We have the first stage as object representation, then we have the second stage as modeling transformation, third stage lighting or coloring, fourth stage is the viewing pipeline which itself consists of 5 sub stages, the viewing transformation, clipping, hidden surface removal projects and transformation and window to viewport transformation. And finally, we have the scan conversion which is the fifth and final stage. Now, we are discussing the first stage, object representation.

(Refer Slide Time: 1:44)



So far what we have discussed, let us have a relook. There are broadly 4 techniques that are popularly used for representing objects, one is point sample rendering, then we have boundary representation. Now, in boundary representation, there are three techniques: mesh representation, parametric representation and implicit representation. In the previous few lectures, we talked about point sample rendering, the boundary representation techniques including mesh representation, parametric representation and implicit representation.

Also, in the previous couple of lectures, we discussed in details one of the very popular parametric representation technique, namely the spline representation. The other techniques include space representation or space partitioning based representation technique, which itself has 3 sub techniques or 3 BSP or binary space partitioning and CSG. Then, we have sweep representation having further techniques namely sweep surface representation and surface of revolution representation.

Apart from these four, there are some other implementation techniques which are application specific, and some advanced techniques such as scene graph, skeleton model and advanced modeling techniques including particle systems, fractal systems and so on. And we have got some idea on various representation techniques in our previous lectures. Today we are going to discuss in details the space partitioning technique for representing objects including its sub techniques.

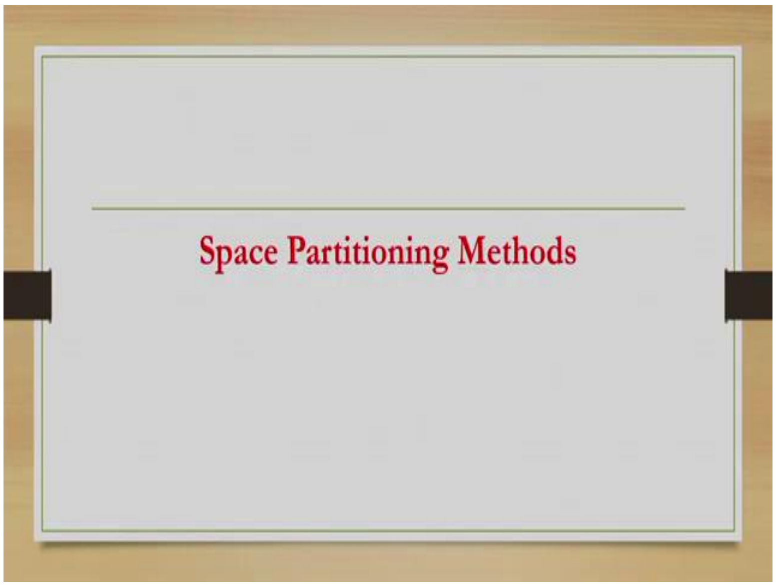
(Refer Slide Time: 3:56)

Recap

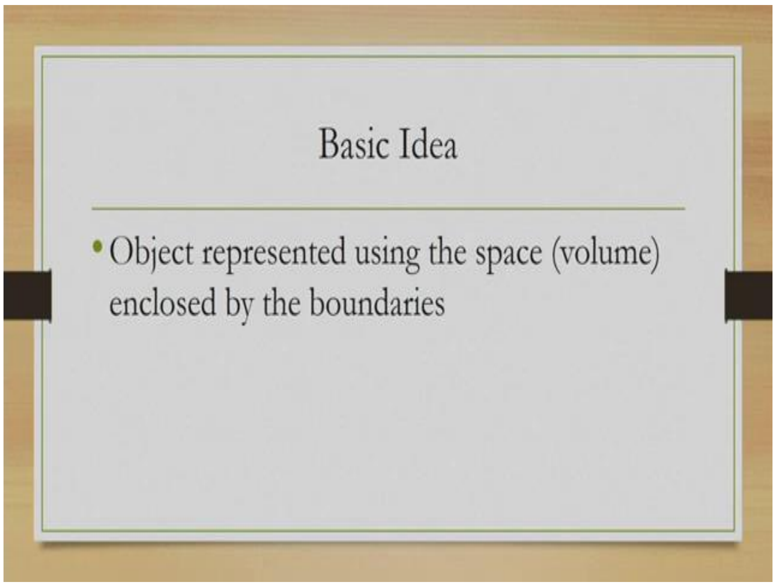
- We learned about boundary representation
- Today, we shall learn **space partitioning methods**

So, that will be the topic of our discussion today, space partitioning methods.

(Refer Slide Time: 4:04)



Space Partitioning Methods



Basic Idea

- Object represented using the space (volume) enclosed by the boundaries

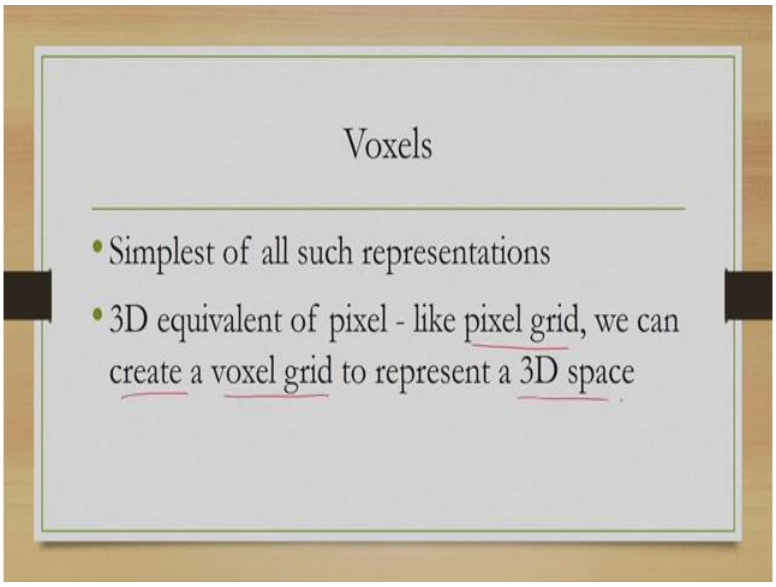
Now, when we talk about space partitioning, what we refer to? As the name suggests, space partitioning refers to the representation of an object in terms of the 3D space that it occupies. The space is defined by its boundaries and we represent the space or the volume that is enclosed by the boundaries rather than the boundaries, which we have seen in the earlier lectures.

(Refer Slide Time: 4:43)

Voxels

Now, there is one important concept in space partitioning techniques, this is called voxels. You have heard of pixels, we mentioned this term earlier. This is the smallest display unit. And on a display screen, we assume that there is a pixel grid. Similarly, voxel is basically the 3D counterpart of the concept of pixel. Voxel is the smallest unit of representation in a 3D space. Any 3D space can be assumed to be having a voxel grid.

(Refer Slide Time: 5:35)

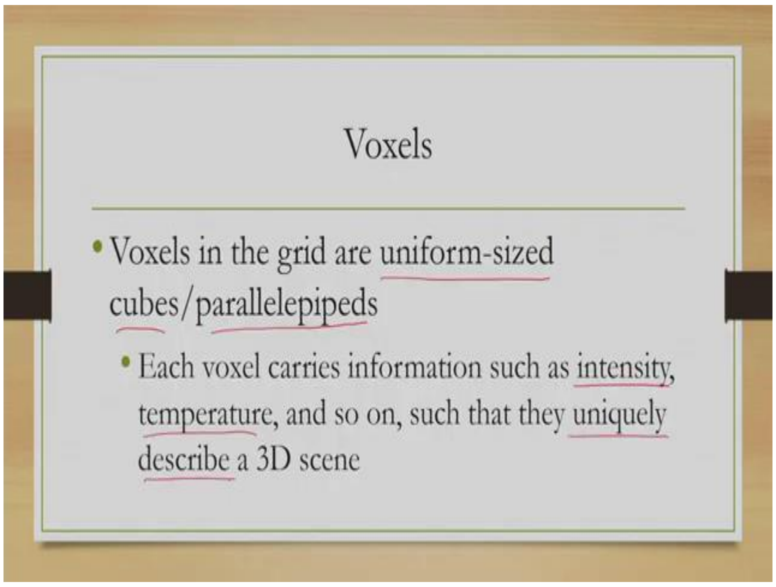


Voxels

- Simplest of all such representations
- 3D equivalent of pixel - like pixel grid, we can create a voxel grid to represent a 3D space

Like a pixel grid, we can create or assume that a voxel grid exists to represent a 3D space. So, pixel grid is for 2D space, voxel grid is for 3D space. And voxel, as maybe obvious, is the simplest way of representing 3D objects.

(Refer Slide Time: 6:12)

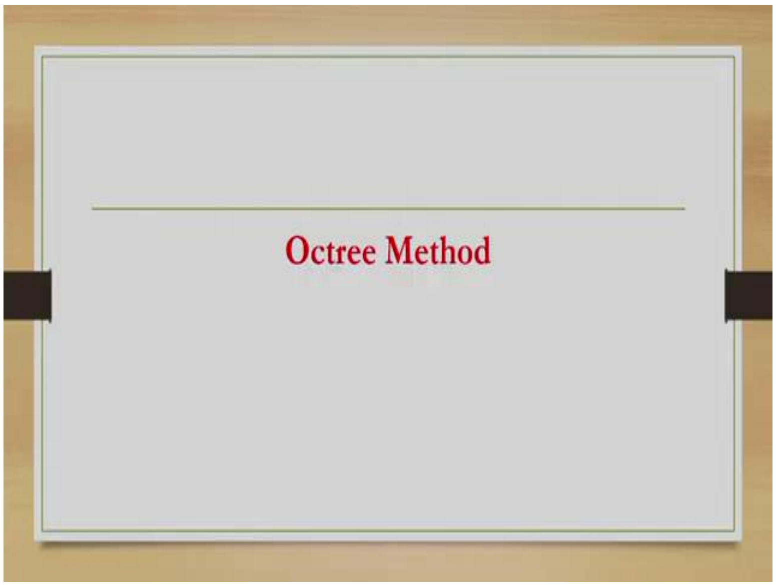


Voxels

- Voxels in the grid are uniform-sized cubes/parallelepipeds
  - Each voxel carries information such as intensity, temperature, and so on, such that they uniquely describe a 3D scene

Now, when we talk about voxels, we are essentially referring to typically uniform sized cubes or parallelepipeds. So, essentially, a grid of uniform sized cubes or parallelepipeds that is our voxel grid. Now, in each grid, each voxel element of the grid typically carries various information. What are those information? The intensity at that particular 3D region, the temperature of that region and so on. And these information actually help in uniquely describing the 3D scene. So, essentially voxels are having attributes which are characteristic information for the particular object in the scene.

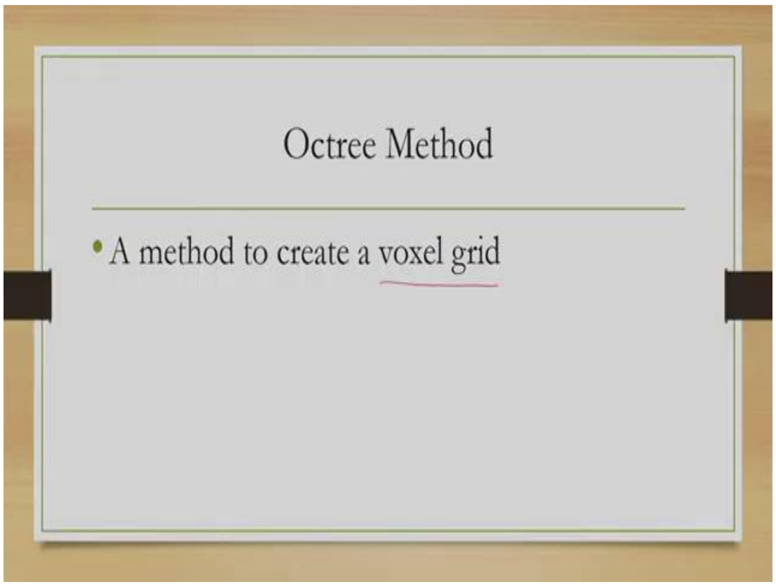
(Refer Slide Time: 7:25)



Octree Method

Using voxels, we can actually apply various techniques to represent objects. One such technique is the Octree method. What is this method? As the name suggests, it is kind of tree representation, now let us try to understand what is this tree.

(Refer Slide Time: 7:51)

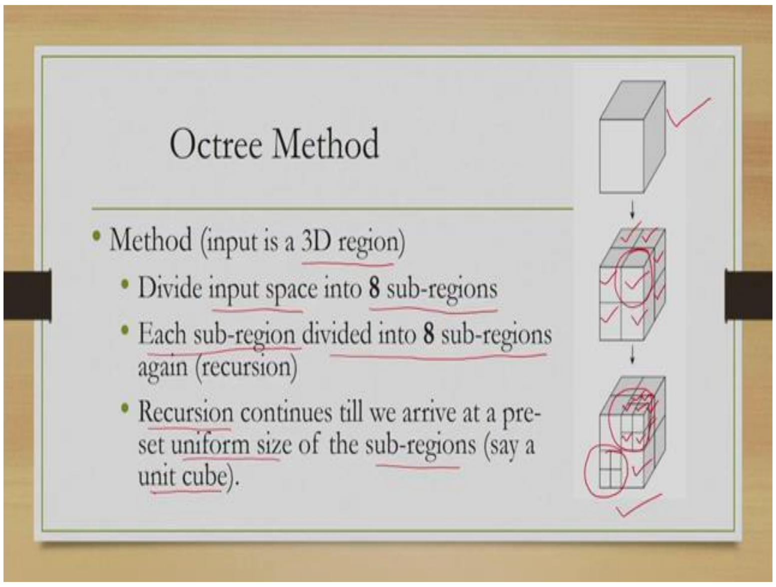


Octree Method

- A method to create a voxel grid

So, essentially it refers to a method to create a voxel grid, but in the form of a tree.

(Refer Slide Time: 7:59)



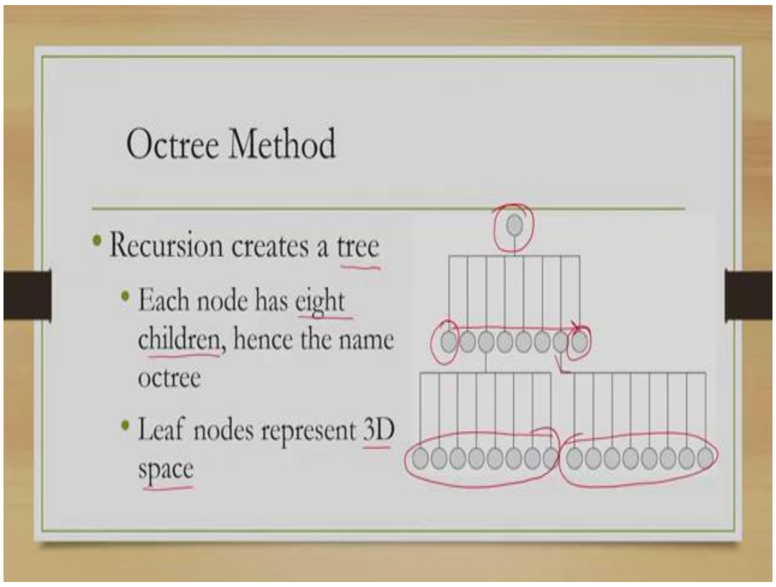
Octree Method

- Method (input is a 3D region)
  - Divide input space into **8** sub-regions
  - Each sub-region divided into **8** sub-regions again (recursion)
  - Recursion continues till we arrive at a pre-set uniform size of the sub-regions (say a unit cube).

So, in this method, the input is a 3D region. Now, we divide this input space into 8 sub regions, then each sub region in turn is again divided into 8 sub sub regions. So, essentially this is a recursive step and this recursion continues till we arrive at some preset uniform size of the sub regions. Now, this size can be user defined, for example, a unit cube. So, when we see that our division leads to unit cube sized regions, then we stop there, an example is shown in this right-hand figure.

So, this is our initial 3D region. Now, as you can see, we have created 8 sub regions from here, this is 1, 2, 3, 4, 5, 6 ,7 and 8, then each of these sub regions are further divided into 8 sub sub regions like here as you can see in this division, again there are 8 sub regions 1, 2, 3, 4, 5, 6, 7, 8. Similarly, here also, we divided and all other sub regions also we can divide.

(Refer Slide Time: 9:50)



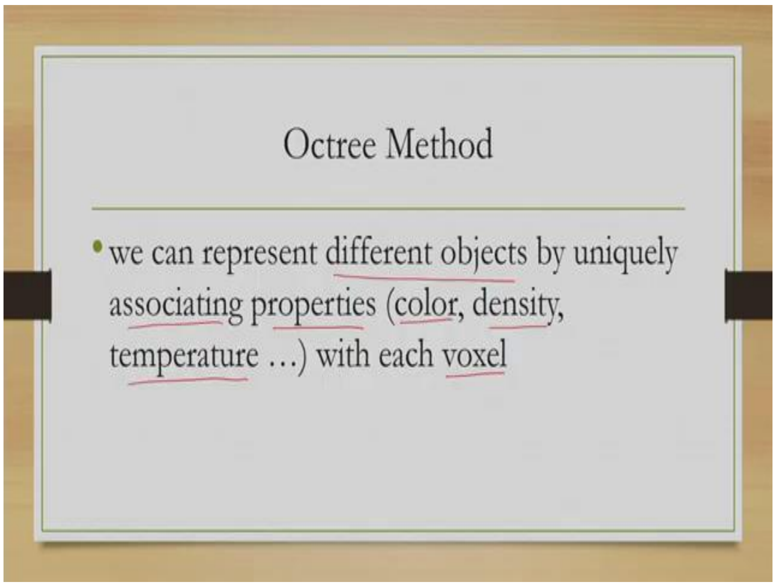
Octree Method

- Recursion creates a tree
  - Each node has eight children, hence the name octree
  - Leaf nodes represent 3D space

Now, what is the output of this recursion? It creates a tree. So, we have a root node. For each root node, we have created 8 child nodes from here to here. Now for each child node, again we are creating 8 child nodes and this process continues. So, essentially this leads to a tree. Since each node has 8 children, we call it octree and the leaf nodes of this tree represent the 3D space. So, all the intermediate nodes are intermediate representations, they do not represent the final object, at the leaf level the final object is represented.

Remember here that along with the space the attributes or the characteristic information is also stored at the leaf node level.

(Refer Slide Time: 10:59)



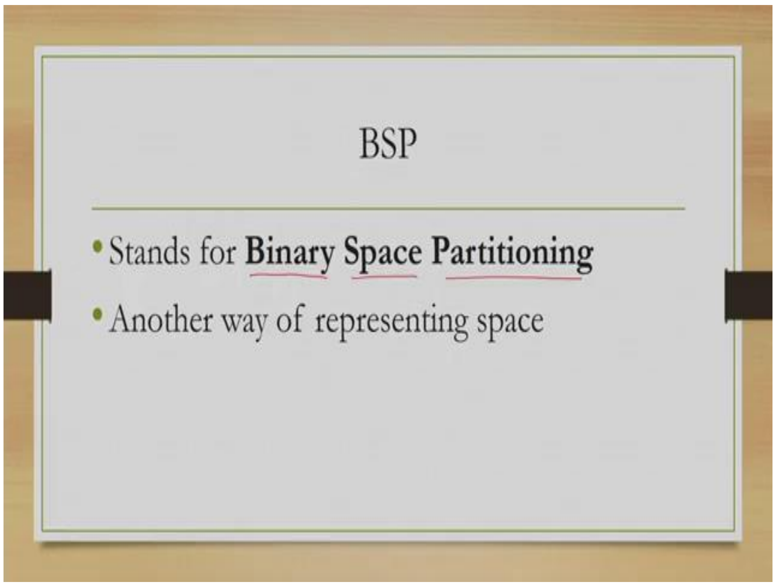
Octree Method

- we can represent different objects by uniquely associating properties (color, density, temperature …) with each voxel

So, in order to represent different objects, we can associate unique properties to those leaf nodes or voxels, the properties such as color, density, temperature and so on, that is the basic idea. So, we are given 3D space, the space we divided into 8 regions and we continue this division, each sub region is divided further into 8 sub sub regions and so on. We continue this division in a recursive manner till we reach the voxel level or the uniform sized cubes. And this process creates a tree, each node in this tree has 8 children, so the tree is called octree.

The leaf level contains the voxels are representation of the 3D object and the characteristic information such as color, density, temperature and so on are associated with each voxel in this tree to uniquely identify objects. Octree is one of the various methods where space is used to represent the objects. Another popular method is called BSP or the BSP method.

(Refer Slide Time: 12:51)

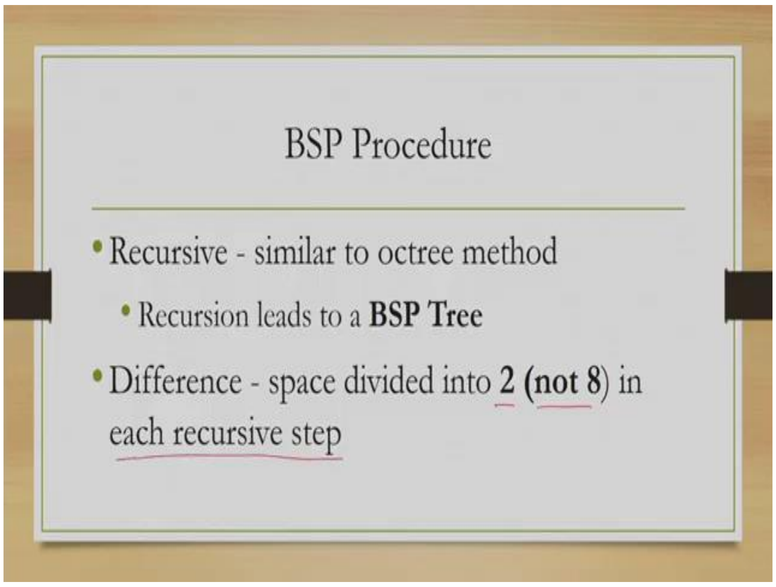


BSP

- Stands for **Binary Space Partitioning**
- Another way of representing space

Now, BSP stands for binary space partitioning and the method is binary space partitioning method. So, what it does? In the octree, we have done some recursive partitioning of space. In the BSP tree, we follow similar recursion, that is we are given a space we divide it into subspace and then divide the subspace again and continue till some condition is met.

(Refer Slide Time: 13:26)

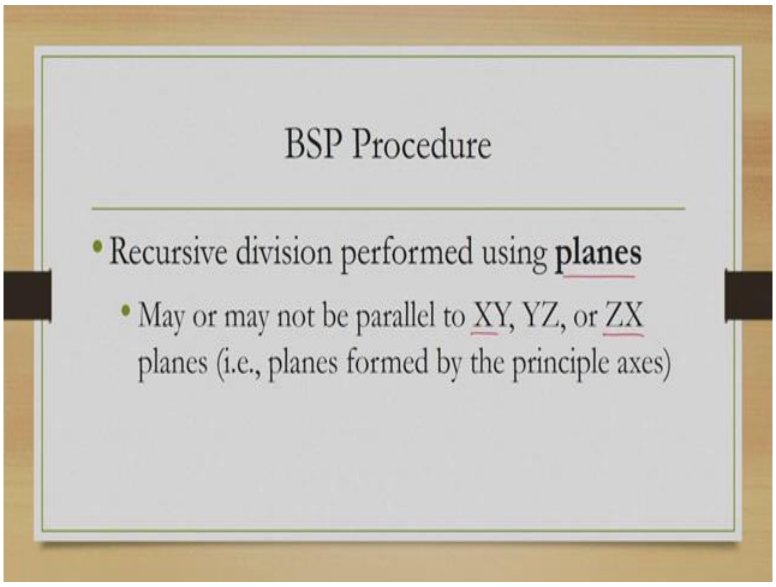


BSP Procedure

- Recursive - similar to octree method
  - Recursion leads to a **BSP Tree**
- Difference - space divided into **2 (not 8)** in each recursive step

However, instead of dividing a space into 8 subspaces like we have done in the case of octree, here what we do we divide it into 2 spaces or subspaces in each recursive step. So, in case of octree we are dividing the region into 8 sub regions, in case of BSP tree we are dividing the region into 2 sub regions, that is why it is called binary space partitioning. Binary or two.

(Refer Slide Time: 14:01)

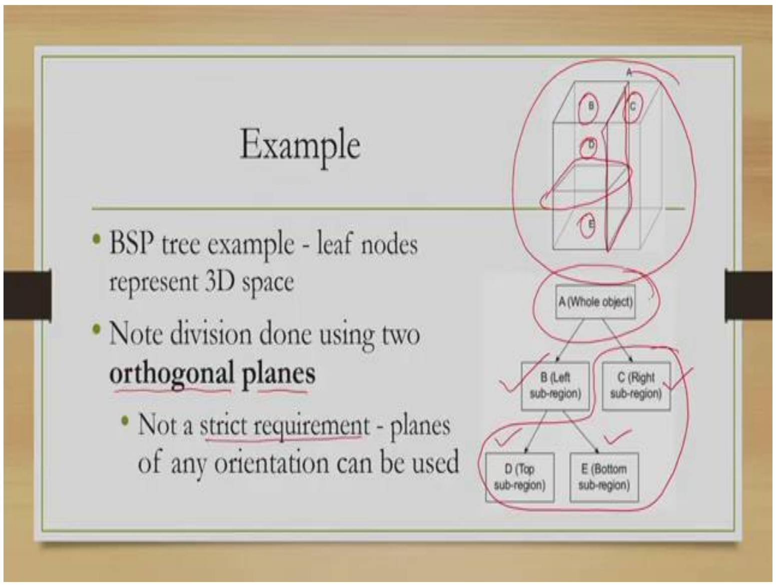


BSP Procedure

- Recursive division performed using **planes**
  - May or may not be parallel to XY, YZ, or ZX planes (i.e., planes formed by the principle axes)

Now, how do we divide? We use planes. So, we are given a region we assume planes are available to divide the region into sub regions. But these planes need not be parallel to the planes formed by the axes XY, YZ or ZX. It can be planes of any orientation. Of course, if we are having parallel to the planes formed by the principal axis then it is easier to implement, otherwise we have to do additional computations.

(Refer Slide Time: 14:49)



Example

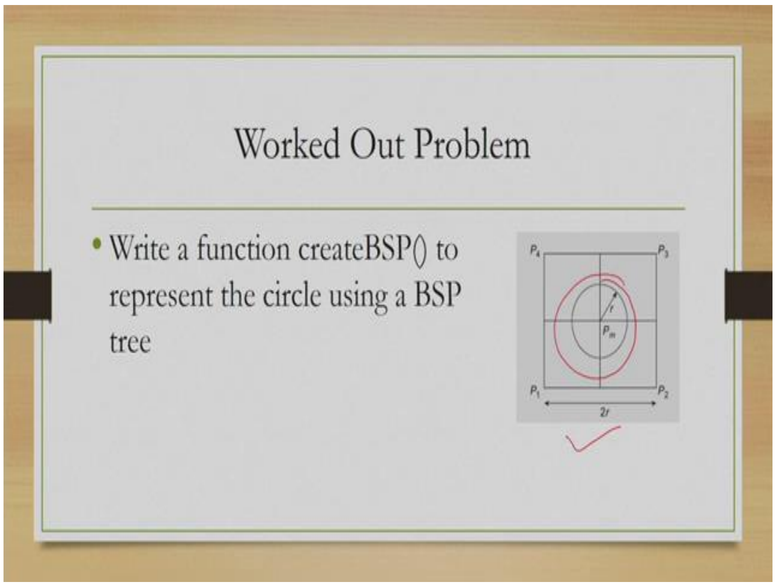
- BSP tree example - leaf nodes represent 3D space
- Note division done using two **orthogonal planes**
  - Not a strict requirement - planes of any orientation can be used

Let us see one example. Suppose we are given this 3D region and we are going to represent it in the form of a BSP tree. So, our root node is the whole object, then we have used a plane this one to divide it into two regions, the left region D and the right region C. Left with respect to the plane and right with respect to the plane, left sub region and right sub region. Then we have used another plane here to divide B into two sub regions D and E. So, then eventually what we got? The object is represented in terms of three sub regions D, E and C.

So, these three at the leaf level represent the object. And again like in case of octree, we can associate unique properties with each of these sub regions to uniquely identify the object. Now, here we have used two planes which are orthogonal to each other, they are orthogonal planes, but that is, as we have already seen earlier, that is not a strict requirement. Planes of any orientation can be used to divide the regions into two sub regions.

So, here instead of these orthogonal planes, we could have used any plane of any orientation to divide it into two regions, two sub regions, that is the most general formulation for BSP tree creation. But, as I said, if we are using planes that are not parallel to the planes formed by the principal axis, then additional computations may be required to process the representation. Let us see one more example for creation of BSP tree, how we can create, how we can write an algorithm for creation of a representation.

(Refer Slide Time: 17:29)



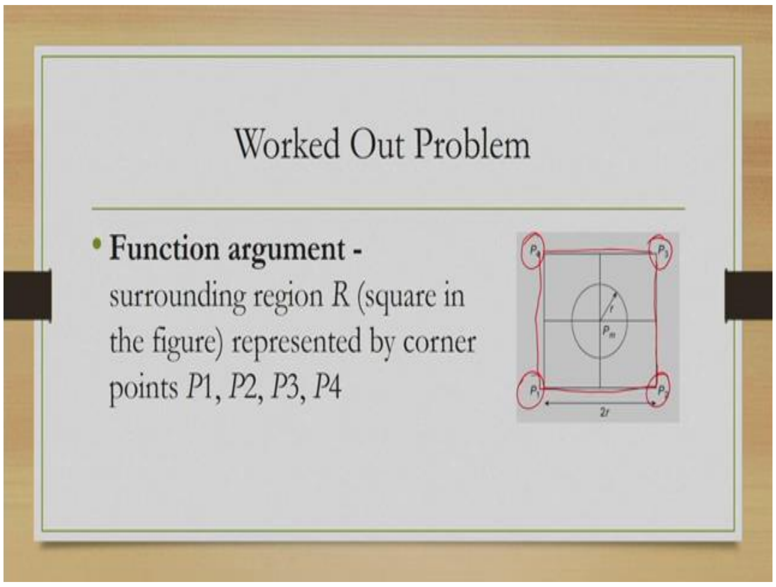
Worked Out Problem

- Write a function createBSP() to represent the circle using a BSP tree

P_4
P_3
r
P_m
P_1
P_2
$2r$

Consider this figure, here we want to represent this circle, this is of course a two dimensional figure, it is not a 3D object, but this 2D object we want to represent this circle which is having the center at Pm and radius r. So, we want to represent it using BSP tree.

(Refer Slide Time: 18:01)



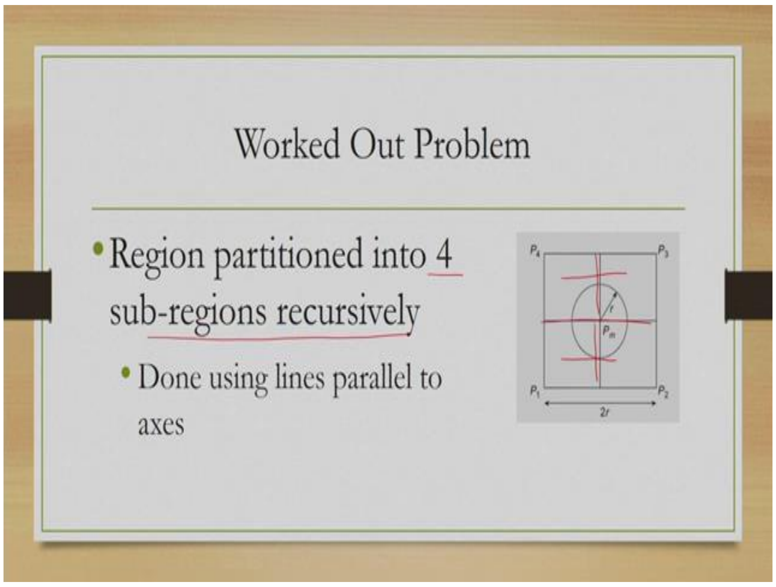
Worked Out Problem

- **Function argument -** surrounding region *R* (square in the figure) represented by corner points *P*1, *P*2, *P*3, *P*4

P_4
P_3
r
P_m
P_1
P_2
$2r$

How we can do that? So it take as input the surrounding region R which is the square represented by the four vertices or the corner points P 1, P 2, P 3 and P 4 and the circle is within this region. So, when we are representing the circle, we are essentially representing this region with some regions that are part of the circle having unique characteristics of the circle. How we can do that?

(Refer Slide Time: 18:46)



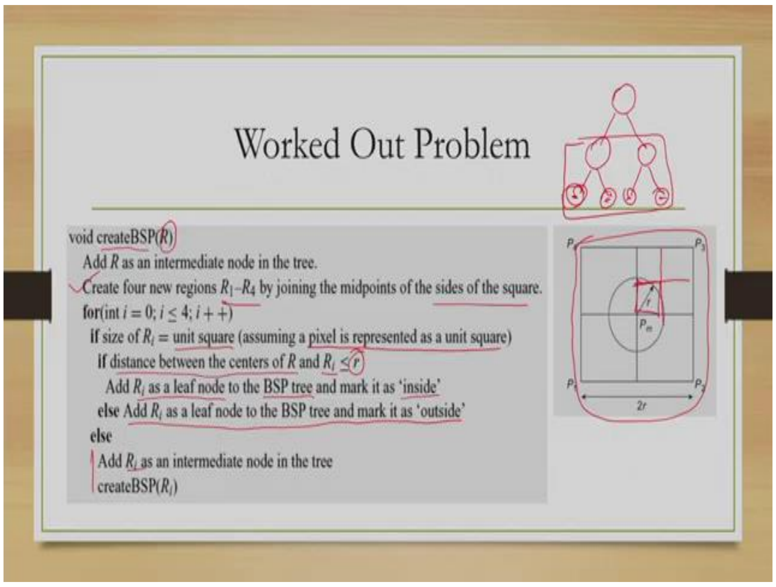
Worked Out Problem

- Region partitioned into 4 sub-regions recursively
  - Done using lines parallel to axes

P_4
P_3
r
P_m
P_1
P_2
$2r$

So, we will partition the region into 4 sub regions. So, first we divide into 2 sub regions, then each of these 2 sub regions we divide into further 2 sub regions and so on. So, just for simplicity we are combining these steps together here and stating that we are dividing it into 4 sub regions. And here we are using lines parallel to the axes.

(Refer Slide Time: 19:25)



Worked Out Problem

```
void createBSP(R)
  Add $R$ as an intermediate node in the tree.
  Create four new regions $R_1$–$R_4$ by joining the midpoints of the sides of the square.
  for(int $i = 0; i \leq 4; i++$)
    If size of $R_i$ = unit square (assuming a pixel is represented as a unit square)
      If distance between the centers of $R$ and $R_i \leq r$
        Add $R_i$ as a leaf node to the BSP tree and mark it as 'inside'
      else Add $R_i$ as a leaf node to the BSP tree and mark it as 'outside'
    else
      Add $R_i$ as an intermediate node in the tree
      createBSP($R_i$)
```

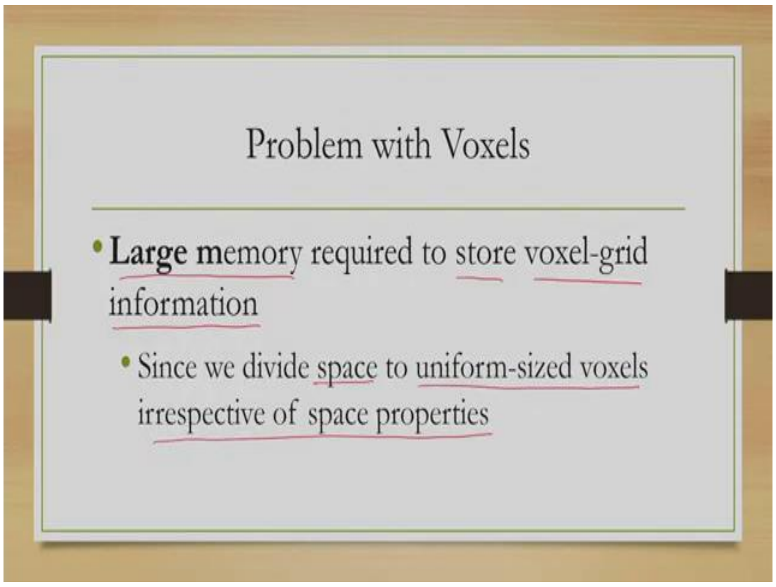
Using that idea, what we can do? We can write an algorithm for a function create to BSP where R is the input region. Now, we will use R to create a node, then from R will create 4 regions by joining the midpoints of the sides of the square, this is same as applying the binary spaced partitioning techniques in multiple steps. Like in this case, we first create 2, then for each of these two create two more and we are actually combining these steps here in this line.

Now, for each of these sub regions here at this leaf node of this figure, if the size of this sub region, suppose this region is divided into 4 sub regions, so we are considering this. Now, if the size of the sub region is a unit square, where we are assuming that a pixel is represented as a unit square. Then we go to the next step that is if distance between the centers of the original region R and the sub region that we are considering currently is less than or equal to the radius of the circle, then this sub region is part of the circle.

So, we add it as a leaf node to the BSP tree and mark it as 'inside'. Otherwise, we mark it as 'outside', although add it as part of BSP tree. Now, if the size is not unit square, that is we have not reached the termination condition of recursion. So, for each node we perform the function again, that is we call the function again recursively mentioned in these 2 steps. So, at the end of it we get a tree, where the leaf nodes represent the bounding region, the original region divided into sub regions.

Now, some of these sub regions are marked as inside this particular object, inside the square. Whereas, others are marked as outside. So, from that tree, we get a representation of the square. So far so good. Now, what is the problem? Is there any problem with this space partitioning approach?

(Refer Slide Time: 23:00)



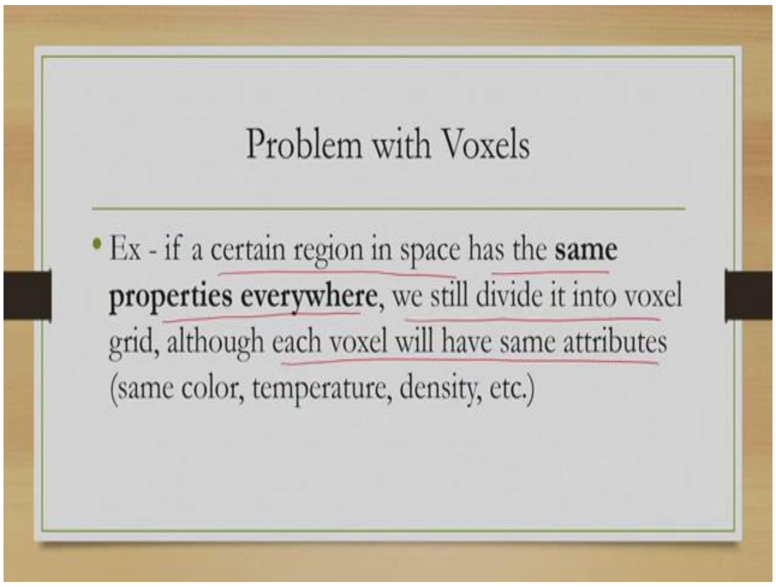
Problem with Voxels

- **Large** memory required to store voxel-grid information
  - Since we divide space to uniform-sized voxels irrespective of space properties

One problem may be very obvious to you by now, that is we require large memory to store these voxel grid information. So, we are creating tree where the leaf nodes represent the grid and if we are partitioning uniformly, then there will be large number of voxels and we require significant amount of memory space to store the voxel information. The problem comes because we are dividing space into uniform sized voxels irrespective of the space properties.

Like we have seen in the earlier example, the region R is a big region, within this, circle occupies a small area, but we are dividing the region into uniform sized sub regions and many sub regions may be outside the circle. So, if you want to represent the circle, it is not necessary to represent those regions which are outside the circle that actually wastes some memory space. So, if we have some method to avoid these wastage, then of course that would be helpful.

(Refer Slide Time: 24:21)

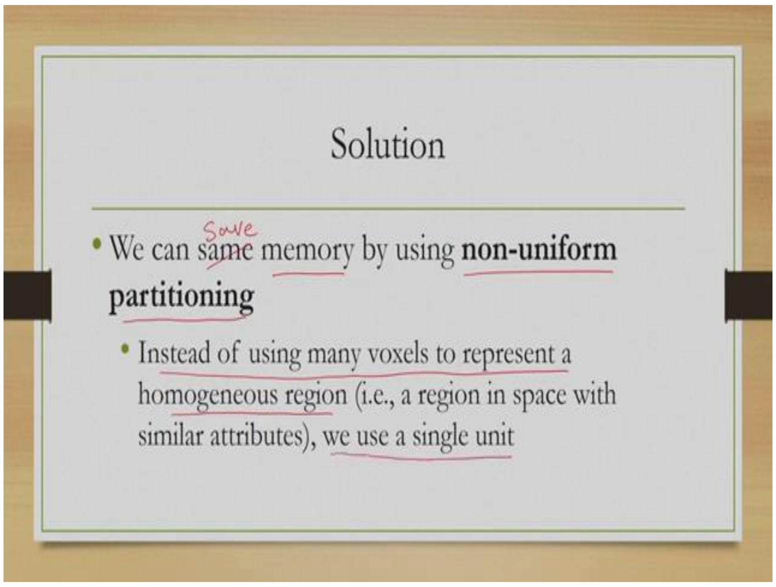


Problem with Voxels

- Ex - if a certain region in space has the **same properties everywhere**, we still divide it into voxel grid, although each voxel will have same attributes (same color, temperature, density, etc.)

So, what do we want? If a certain region in space has the same properties everywhere, then ideally we should not divide it further. Because even if we divide whatever we get, we will get the same property. Instead, what we do, we still divide it into voxels, although each voxel contain the same attributes, because the broader region or the broader region in space has the same property everywhere.

(Refer Slide Time: 24:59)

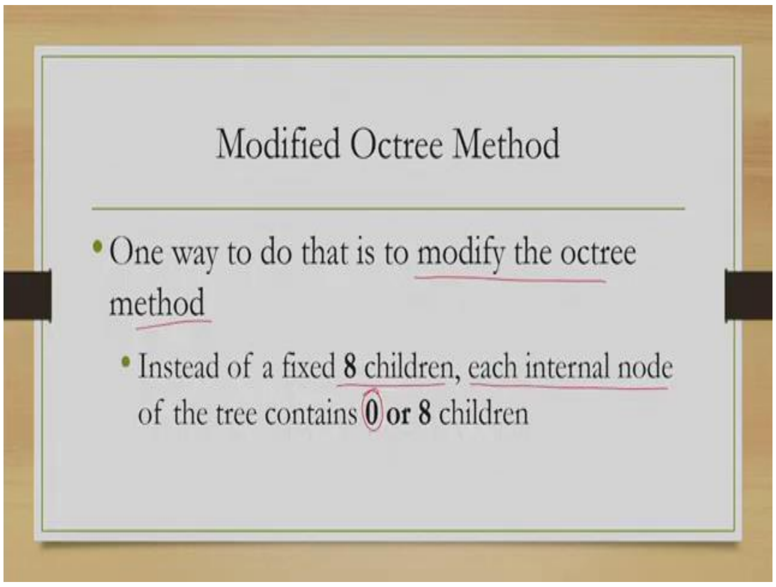


Solution

- We can ~~same~~ save memory by using **non-uniform partitioning**
  - Instead of using many voxels to represent a homogeneous region (i.e., a region in space with similar attributes), we use a single unit

So, we can actually save, this is there is a typo here, it is safe, we can save memory by using this idea of non-uniform partitioning. So, earlier we are talking of partitioning a region into uniform sized voxels. Now, we are talking of non-uniform sized space partitioning. What is that? That is, instead of using many voxels to represent a homogeneous region that is a region where property is same everywhere we use a single unit. So, we do not divide that region further, instead the whole region is represented as a single unit.

(Refer Slide Time: 25:47)



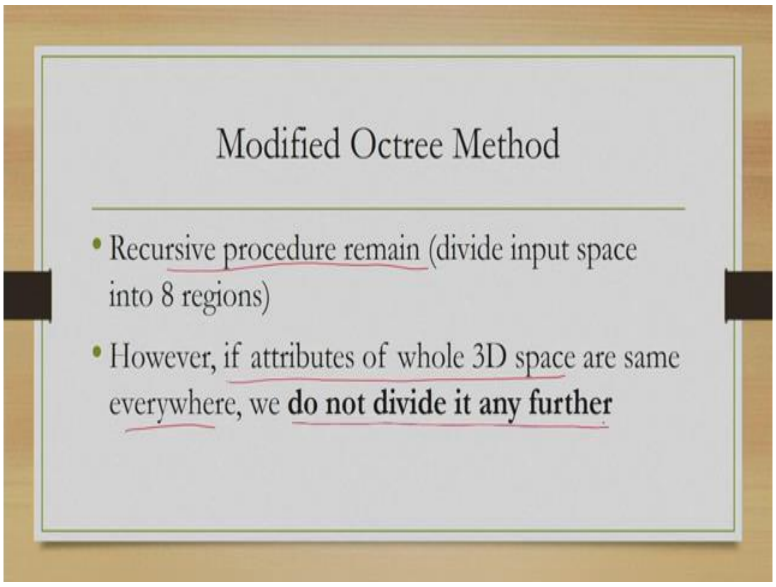
Modified Octree Method

- One way to do that is to modify the octree method
  - Instead of a fixed **8** children, each internal node of the tree contains **0 or 8** children

How we can do that? One way is modify the octree method. Now, earlier we were having fixed 8 children for each node because we were dividing the region into 8 sub regions irrespective of the property of the region. Now, what we can do? Either we divide or we do not divide. So, if one sub region is having same property everywhere, then we do not divide it. So, that node will not have any children or 0 children. But if it is not the case, then we divide it into 8 sub regions or 8 children.

So now, in the revised octree method, what we will have either 0 or 8 children for each node, which was not the case earlier. So, earlier we were getting 8 children for each node, intermediate node, now we will get either 0 or 8 children for each intermediate node depending on the property of that space represented by the node.

(Refer Slide Time: 26:58)

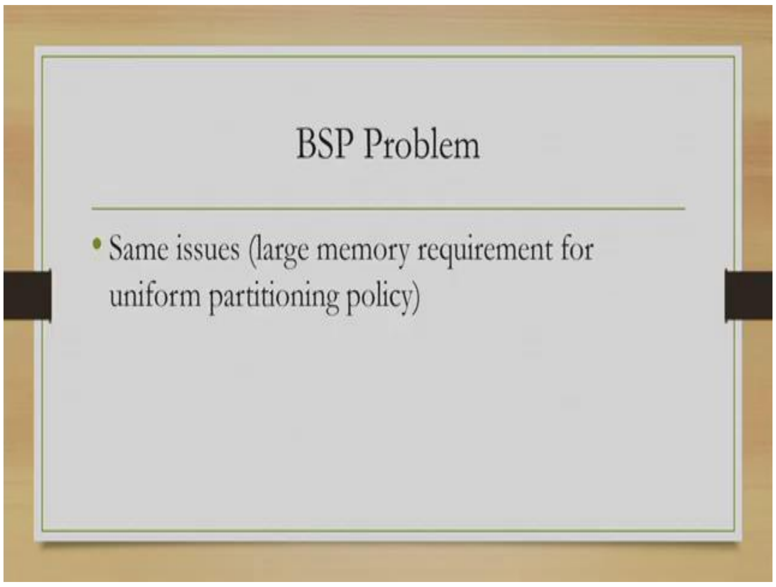


Modified Octree Method

- Recursive procedure remain (divide input space into 8 regions)
- However, if attributes of whole 3D space are same everywhere, we **do not divide it any further**

The recursion procedure, of course, will remain, the recursion procedure will remain the same. But at each step of the recursion, we will check if attributes of a space is same everywhere. If that is the case, then we do not divide it any further. So, we do not go for recursive division again for that particular region.

(Refer Slide Time: 27:35)

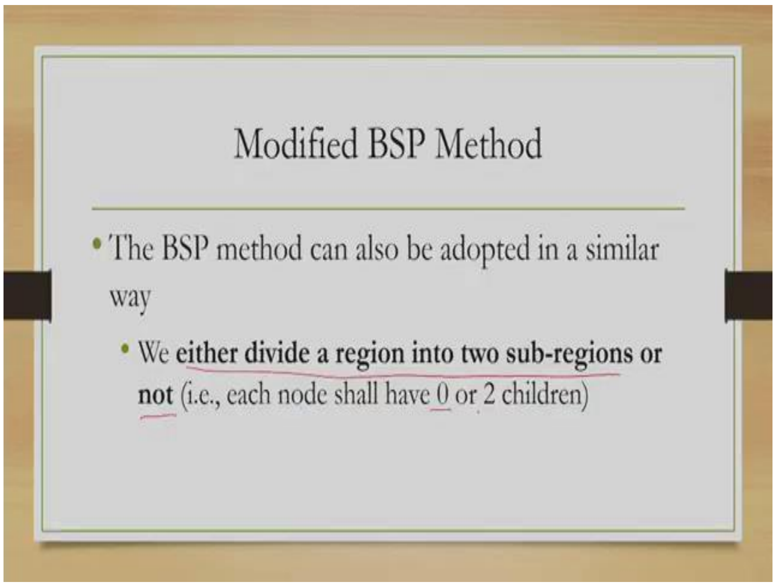


BSP Problem

- Same issues (large memory requirement for uniform partitioning policy)

BSP representation also suffers from the same problem if we are going for uniform partitioning.

(Refer Slide Time: 27:54)



Modified BSP Method

- The BSP method can also be adopted in a similar way
  - We **either divide a region into two sub-regions or not** (i.e., each node shall have 0 or 2 children)

And to avoid that, we can go for a revised method or a refined method where we can either divided region into 2 sub regions if the sub regions are having different properties at different places or we do not divide. So, the intermediate nodes will have either 0 or 2 children similar to the revised octree method. So, what we have learned is that the basic unit of representation will be voxel. Now, using voxel we can divide a given 3D space to represent the object contained in that space in two ways, one is uniform division.

Now, when we are going for uniform division, we will get one particular type of tree if we are following octree method or one particular type of tree if we are following the BSP method. In the octree method, if we are going for uniform division, then we will get 8 children for each node. In the BSP method, we will get two children for each node. Now, if the region which we are dividing is having the same attribute or same properties everywhere then it will be a wastage of memory to divide that region into sub regions and store the information.

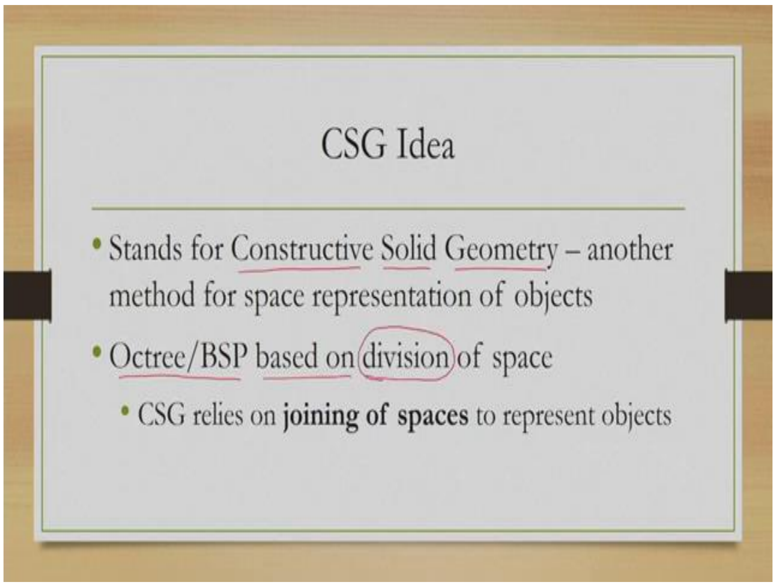
Instead, we need not divide it any further. So, in the case of octree method, we modify it for non-uniform partitioning by imposing the fact that a node can have either 0 or 8 children. Similarly, in a revised BSP method, we can modify by imposing the fact that each intermediate node can have 0 or 2 children.

(Refer Slide Time: 29:56)

CSG

There is another space partitioning method known as CSG. What it means?

(Refer Slide Time: 30:07)

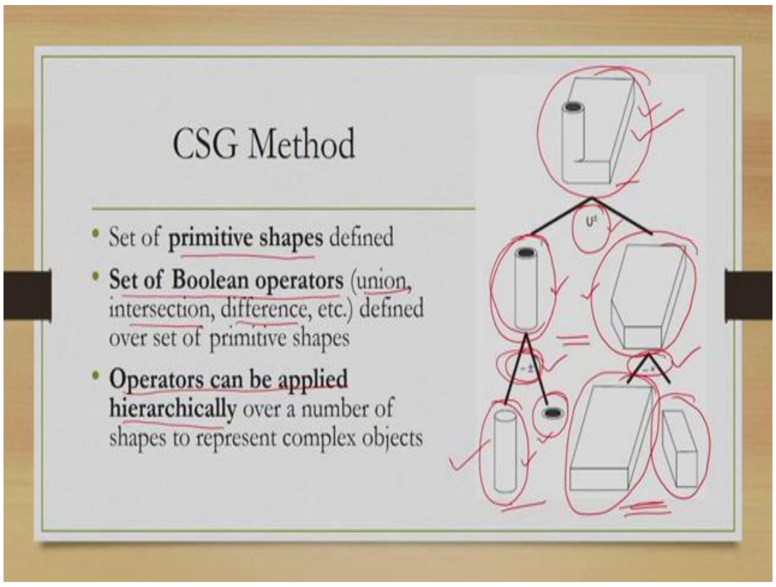


CSG Idea

- Stands for Constructive Solid Geometry – another method for space representation of objects
- Octree/BSP based on division of space
  - CSG relies on **joining of spaces** to represent objects

It stands for Constructive Solid Geometry. So, this is another method for space representation of objects. Now, in case of octree or BSP what we have seen? We have seen that these representations are based on division of space. So, we are given a space and we are dividing it into subspaces. In comparison to that, in case of CSG what we have? It is actually based on joining up spaces, just the opposite of what we do in BSP or octree methods. So, in case of CSG we rely on joining of spaces to represent objects. Let us try to understand this with respect to an example.

(Refer Slide Time: 31:00)



CSG Method

- Set of **primitive shapes** defined
- **Set of Boolean operators** (union, intersection, difference, etc.) defined over set of primitive shapes
- **Operators can be applied hierarchically** over a number of shapes to represent complex objects

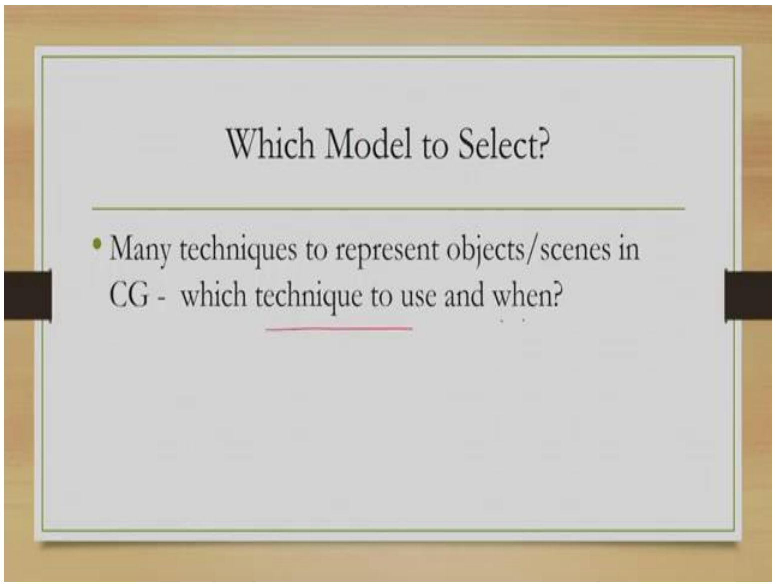
Consider this object, this looks pretty complex. However, when we are using CSG or constructive solid geometry method, we can represent it as a union of some other shapes. So, we start here at this level, here we have this shape, this shape, this shape and this shape, so 4 shapes are there. Now, we combine these 2 shapes to get one shape and we combine these 2 shapes here to get this shape. Now, these 2 shapes are then further combined to get this overall shape.

So, here we start with set of primitive shapes or basic shapes, then we apply a set of Boolean operators namely union, intersection, difference etc. which are defined over this set of primitive shapes to get newer shapes. Like here on these 2 shapes, we applied a Boolean operator to get a new shape and this process continues till we get the desired shape here. So, the operators can be applied hierarchically. So, from this level we reach to this level, from this level we reach to this level, at each level we apply the Boolean operators.

So that is a hierarchical application of operators. So, in other words, we have a set of primitive shapes defined and a set of Boolean operators on those shapes also defined. Now, we apply those operators on the shapes to get new shapes and we apply those operators hierarchically till we get the desired shape, till we are able to represent the desired shape. So, what is the representation? Representation is essentially the primitive shapes and the Boolean operators applied in a hierarchical manner, so that is the representation.

So, that is in summary, what this constructive solid geometry is all about. Now, let us summarize what we have learned so far.

(Refer Slide Time: 34:07)

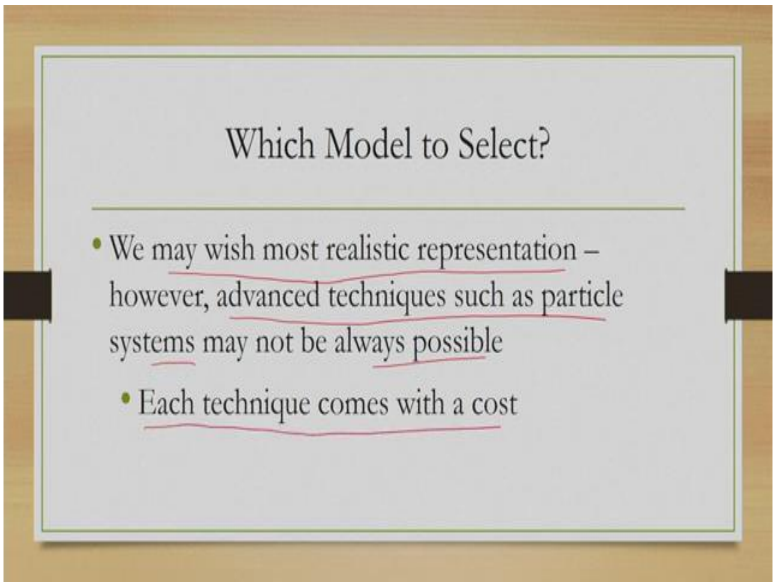


Which Model to Select?

- Many techniques to represent objects/scenes in CG - which technique to use and when?

Today and in previous few lectures we have learned various representation techniques, which is the first stage of the pipeline. Boundary presentation including splines, space representation and also got overview of other representations. Now, there are many techniques as we have seen so far. One question is which technique to use and when? That is a question that always confronts a graphic system developer. Which technique should be used and in which situation? What guides that decision making process?

(Refer Slide Time: 34:55)

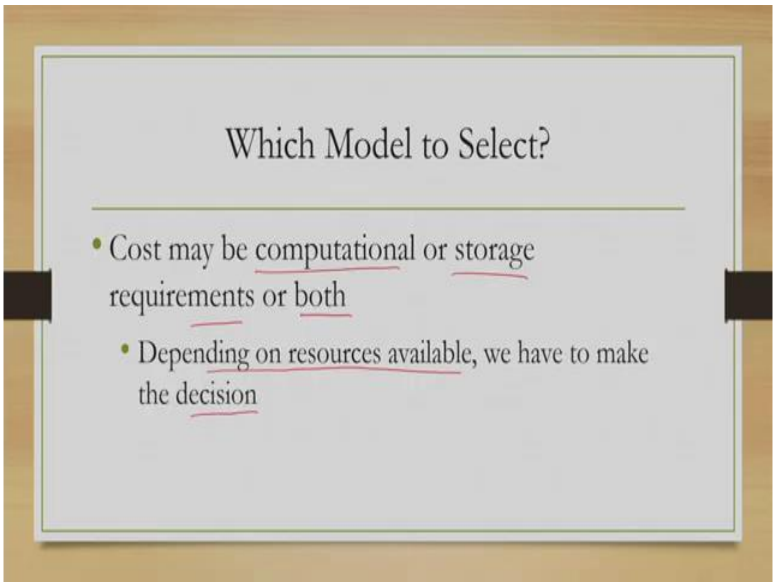


Which Model to Select?

- We may wish most realistic representation – however, advanced techniques such as particle systems may not be always possible
  - Each technique comes with a cost

Now, we may wish to have most realistic representation. Clearly, advanced techniques would be useful but advanced techniques such as particle systems may not always be possible because they require lots of computational resources which may not be available in all graphic systems. So, each technique comes with a cost and there is a tradeoff, which technique to use in which situation.

(Refer Slide Time: 35:26)

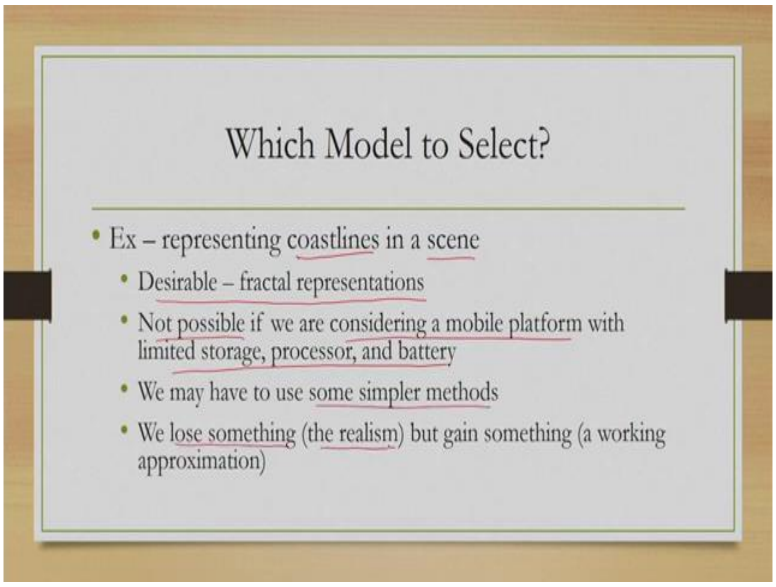


Which Model to Select?

- Cost may be computational or storage requirements or both
  - Depending on resources available, we have to make the decision

Now, this cost may be computational or storage requirement or both and depending on the resource available, we have to make the decision. If we are having very less resource then we cannot think of advanced modeling techniques, then we may have to lose some realism and have to settle for some less realistic scenes or images.

(Refer Slide Time: 36:07)



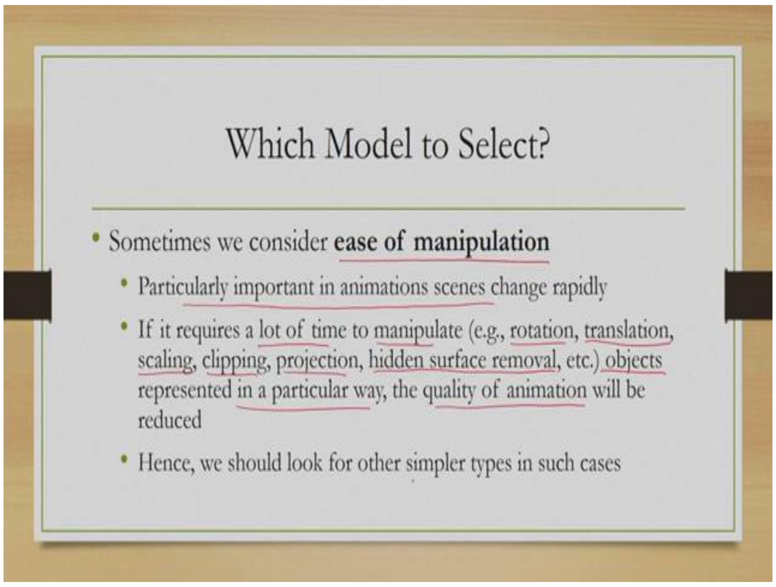
Which Model to Select?

- Ex – representing coastlines in a scene
  - Desirable – fractal representations
  - Not possible if we are considering a mobile platform with limited storage, processor, and battery
  - We may have to use some simpler methods
  - We lose something (the realism) but gain something (a working approximation)

For example, suppose we want to represent coastlines in a scene. So, what is desirable? Coastlines are typically self repeating structures and fractal representation is very good for such shapes. So, we ideally should use fractal representation to have a realistic coastline displayed. But it may not be possible if we are considering a mobile platform with limited storage processor and battery because fractal representation has additional computational cost which may not be supported in low end mobile devices.

So, in that case, we may have to use some simpler method. So, we are losing here something, the realistic effect, but gain something which is a working approximation. So, such trade-offs are very much part of the decision making process, balancing those trade-offs.

(Refer Slide Time: 37:18)



Which Model to Select?

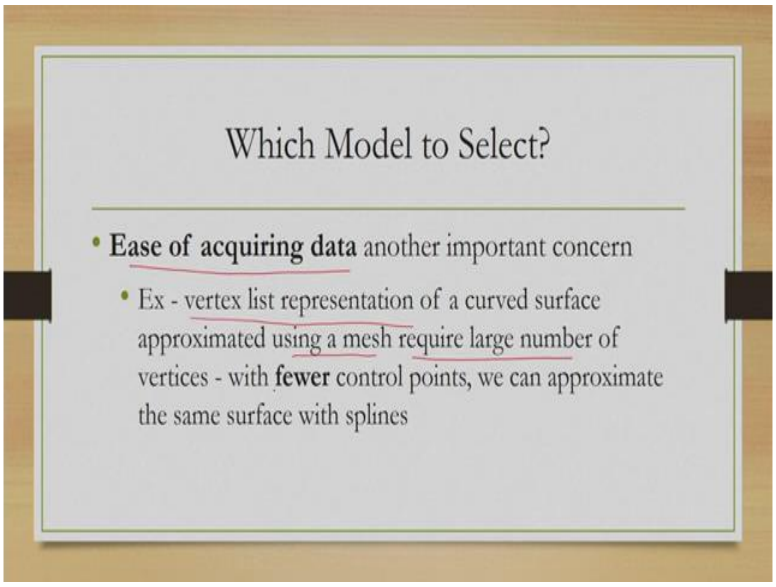
- Sometimes we consider **ease of manipulation**
  - Particularly important in animations scenes change rapidly
  - If it requires a lot of time to manipulate (e.g., rotation, translation, scaling, clipping, projection, hidden surface removal, etc.) objects represented in a particular way, the quality of animation will be reduced
  - Hence, we should look for other simpler types in such cases

Another consideration maybe ease of manipulation. So, when we are creating animation, this consideration comes handy and it becomes important. Now, as we said each representation has its own methods, own algorithms, own process. Subsequent stages of the pipeline needs to process these representations to carry out the operations that are part of the subsequent stages. So, that requires manipulation of the representations.

Now, if it requires a lot of time to manipulate, for example, rotate an object, shift 10 objects horizontally or vertically, scale up or down an object or clip, project or perform better in surface removal, etc. these are part of other stages of the pipeline. We will discuss about the stages in details later. Now, if a lot of time is required to perform these operations, for objects that are represented in a particular way, the quality of animation may be reduced. So, in such cases, we should look for simpler types.

So, this is another consideration that should be kept in mind while choosing a particular model of representation, that is ease of manipulation. So, if we are having low end devices, low end graphics systems, then it is not advisable to go for advanced representation techniques which require lots of manipulations later, particularly in the context of animation.

(Refer Slide Time: 39:18)



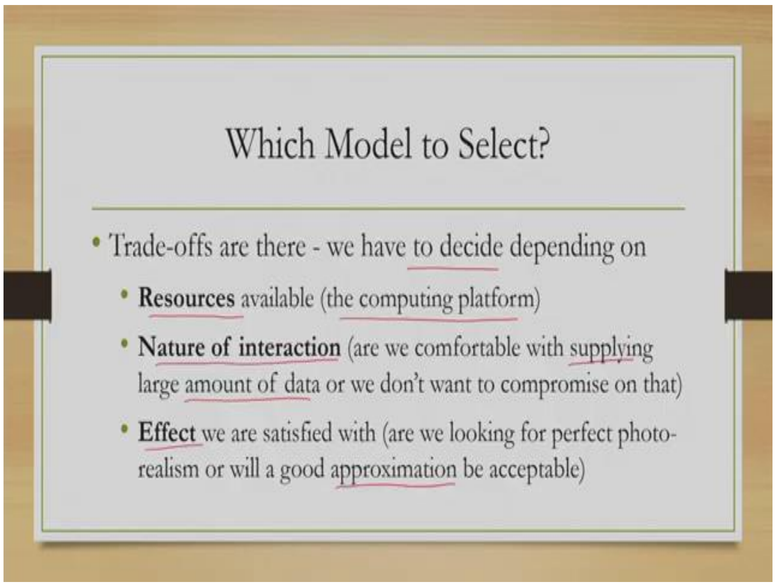
Which Model to Select?

- **Ease of acquiring data** another important concern
  - Ex - vertex list representation of a curved surface approximated using a mesh require large number of vertices - with **fewer** control points, we can approximate the same surface with splines

Third consideration is ease of acquiring data. For example, vertex list representation for a curved surface using a mesh require large number of vertices to be specified. Now, we consider spline representation. In that case, we require fewer control points to represent the same curve. So, clearly here, representing a curve using mesh involves more effort for acquiring data, whereas representing the same curve with a spline involves less effort to acquire data.

So, sometimes these ease of acquiring data can be a deciding factor. Depending on the designer of the graphic system, a particular method can be choosing where the data can be acquired easily.

(Refer Slide Time: 40:22)

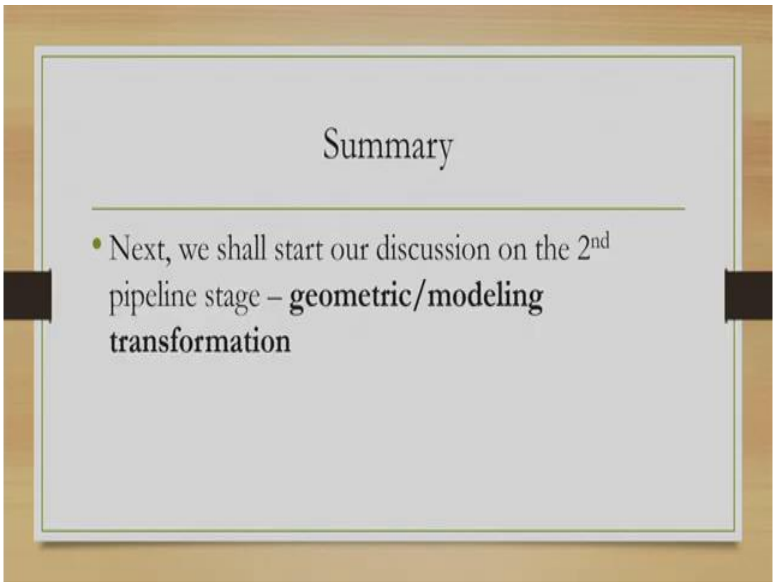


Which Model to Select?

- Trade-offs are there - we have to decide depending on
  - **Resources** available (the computing platform)
  - **Nature of interaction** (are we comfortable with supplying large amount of data or we don't want to compromise on that)
  - **Effect** we are satisfied with (are we looking for perfect photo-realism or will a good approximation be acceptable)

So, there are several trade-offs we have to balance and we have to decide what are those trade-offs resources available, nature of interactions, resources in terms of computing platform, nature of interaction in terms of our level of comfort with supplying large amount of data and also effect, whether we want a very realistic effect or we are looking for some approximation considering the lack of resources.

(Refer Slide Time: 41:07)



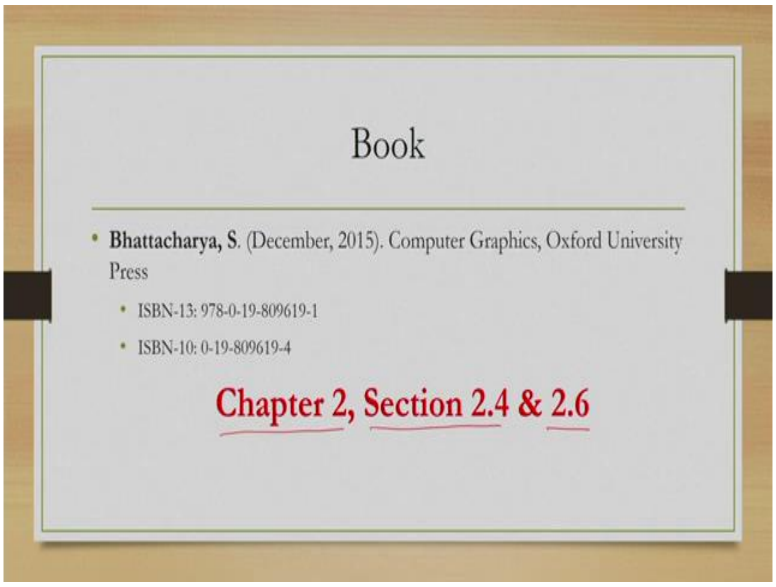
Summary

- Next, we shall start our discussion on the 2nd pipeline stage – **geometric/modeling transformation**

So, depending on these broad 3 considerations, we have to choose a particular modeling technique.

So, with this, we have come to an end to our discussion on the first stage of the graphics pipeline, that is object representation. In the next lecture, we will start our discussion on the second stage of the pipeline, namely, modeling or geometric transformations.

(Refer Slide Time: 41:41)



Book

- **Bhattacharya, S.** (December, 2015). Computer Graphics, Oxford University Press
  - ISBN-13: 978-0-19-809619-1
  - ISBN-10: 0-19-809619-4

Chapter 2, Section 2.4 & 2.6

Whatever I have discussed today can be found in this book. You are advised to go through chapter 2, section 2.4 and 2.6 to get more details on the topics that I have covered today. Thank you and goodbye.