COS 344: L8 Chapter 12: The Graphics Pipeline

Cobus Redelinghuys

University of Pretoria

14/03/2024

Chapter 12: The Graphics Pipeline Introduction

- Data structures are extensively used in computer graphics.
- Today we will discuss the following data structures on a high level:
  - Mesh data structures
  - Spatial data structures
  - Scene graphs
- Note this will be a very high level discussion on the data structures.
- You will only be expected to know where and how they can be used.
- You may use these data structures in your practicals and the homework assignment.

- Before we get to the data structures, let us first discuss the different ways OpenGL can draw primitives.
- Note that not all of the methods are explicitly covered in the textbook, although Section 12.1.3 covers a part of this.
- OpenGL defined a set of primitive rendering modes, in which we can define how OpenGL must draw the vertices that are passed to it.
- The three main categories:
  - Point
  - Line
  - Triangle
- Assume for now that the vertices are stored in a vertex array.
  - Next week we will properly define this.
- `https://en.wikibooks.org/wiki/OpenGL_Programming/GLStart/Tut3`
  - Some of these methods have been deprecated since OpenGL 3.0 and removed since OpenGL 3.1.

Points

- All the vertices in the vertex array are rendered as points.

https://upload.wikimedia.org/wikipedia/commons/b/b2/Glstart_tut3_4.jpg

Lines

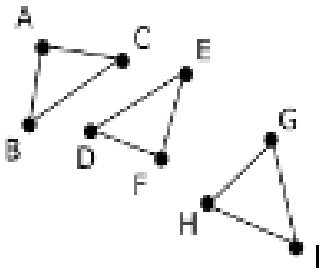
All the vertices in the vertex array are rendered as lines.

- GL_Lines
  - Each pair of vertices represents a line.
  - Say a line is formed between points **a** and **b**, then the next line will be formed between **c** and **d**.
- GL_Line_Strip
  - The vertices are connected to make a loop, excluding connecting the first and last vertex.
- GL_Line_Loop
  - The vertices are connected to make a loop, including the first and last vertex.

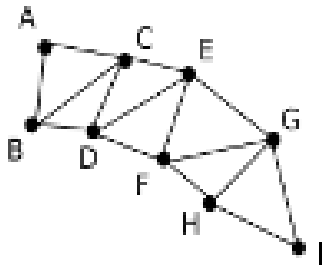
Triangles

All the vertices in the vertex array, are rendered as triangles.

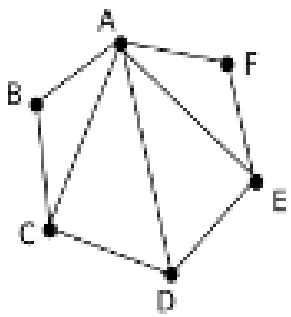
- GL_Triangles
  - A tuple of vertices from the array is drawn as a triangle.
  - Say a triangle is formed with points **a**, **b**, and **c**, then the next triangle will be formed using **d**, **e**, and **f**.
- GL_Triangle_Strip
  - Triangles are formed in a strip.
  - Say a triangle is formed with points **a**, **b**, and **c**, then the next triangle will be formed using **b**, **c**, and **d**.
- GL_Triangle_Fan
  - Triangles are formed in a fan.
  - Say a triangle is formed with points **a**, **b**, and **c**, then the next triangle will be formed using **a**, **c**, and **d**.

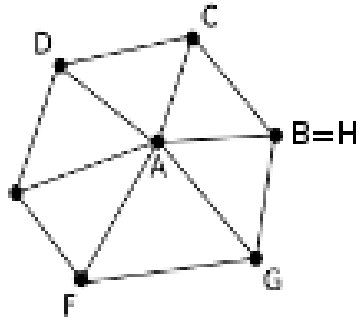


GL_TRIANGLES

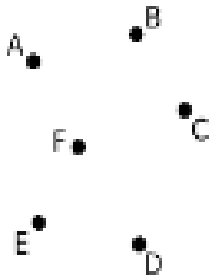


GL_TRIANGLE_STRIP

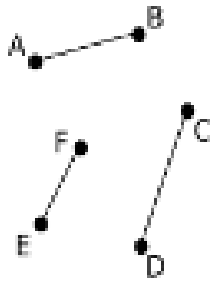




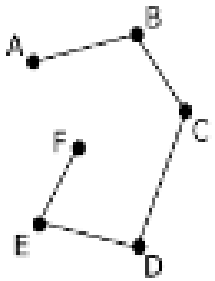
GL_TRINAGLE_FAN



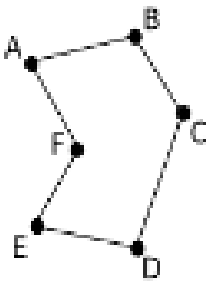
GL_POINTS



GL_LINES



GL_LINE_STRIP



GL_LINE_LOOP

Test type question. Given a set of points, apply the primitive drawing method, or combine the points using the minimum amount of triangles to cover the entire polygon.

Section 12.1: Triangle meshes

- Triangle meshes:
  - Complexes of triangles with a shared vertex.
- Other names:
  - Triangular meshes
  - Triangle meshes
  - Trinagular irregular networks
- Triangle meshes are generally used to represent surfaces.
- Triangle meshes can also be handled more efficiently than a set of triangles.
- Each vertex can store a set of attributes, which is then interpolated as discussed in L7.

Section 12.1.1: Mesh Topology

- Mesh topology:
  - The way triangles connect together, without regard for the vertex positions.
- Manifold restriction
  - The mesh should be water tight.
  - i.e. there should be no gaps in the mesh.
  - Simplest but most restrictive restriction.
  - Following simplified properties, define the manifold restriction
    1. Every edge is shared by exactly two triangles.
    2. Every vertex has a single, complete loop of triangles around it.
  - Note, you do not need to know the mathematical reason behind manifold.

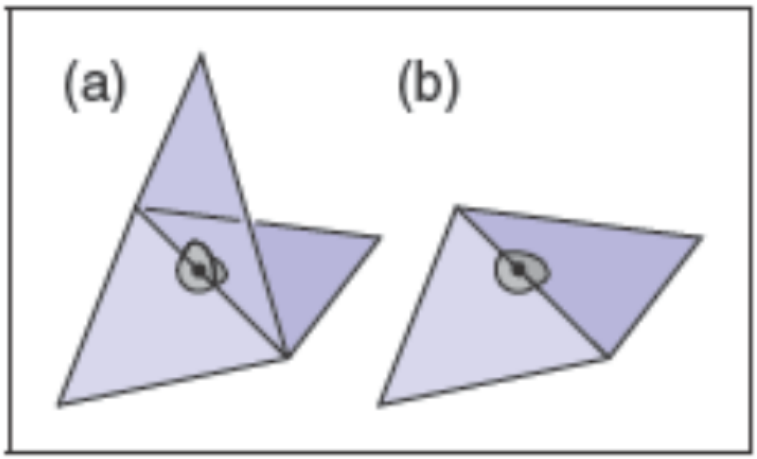


Figure 12.1. Non-manifold (a) and manifold (b) interior edges.

Here, (a) is non-manifold due to the "fin" sticking out, which violates property 1.

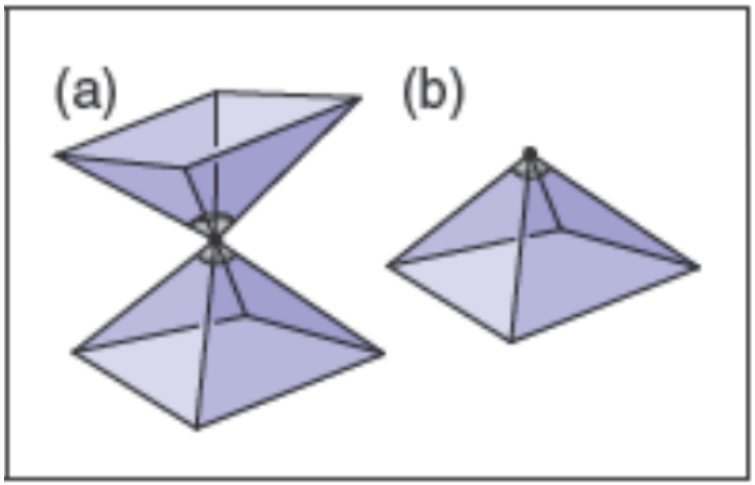


Figure 12.2. Non-manifold (a) and manifold (b) interior vertices.

Here, (a) is non-manifold due to two triangle loops for the centre vertex, which violates property 2.

- Manifold restriction can be relaxed to allow boundaries in the surface.
  - Where would this be useful?
- Manifold's restrictions are relaxed to the following:
  1. Every edge is used by either one or two triangles.
  2. Every vertex connects to a single edge-connected set of triangles.

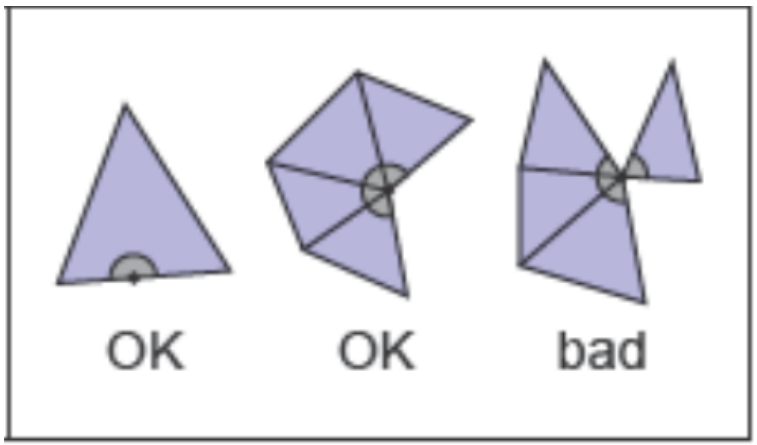


Figure 12.3. Conditions at the edge of a manifold with

The last example is "bad" due to the centre vertex having two sets of triangles.

Section 12.1.2: Indexed Mesh Storage

- Let us assume we store all the corners, of all of the triangles, in a triangular mesh, in an array. What could be the possible problem?

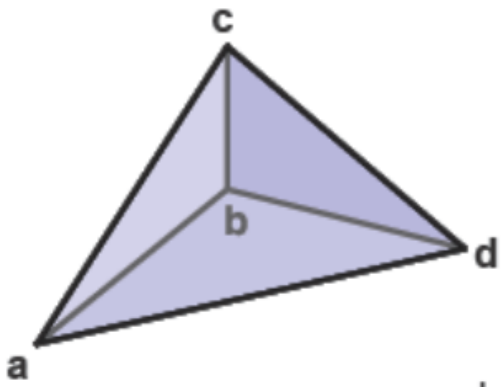
Section 12.1.2: Indexed Mesh Storage

- Let us assume we store all the corners, of all of the triangles, in a triangular mesh, in an array. What could be the possible problem?
  - Duplication as repeated vertices are stored numerous times.
- Solution?

Section 12.1.2: Indexed Mesh Storage

- Let us assume we store all the corners, of all of the triangles, in a triangular mesh, in an array. What could be the possible problem?
  - Duplication as repeated vertices are stored numerous times.
- Solution?
  - Store the unique vertices in one array.
  - Store the indexes of the vertices used to create the triangle in another array.
- Storage calculation:
  - If the mesh has n_v vertices and n_t triangles.
  - Also assume that the storage size of the data types used to store the components of the vertices and the index for the triangles, are the same.

  $triangles = 9n_t$ units $\qquad$ $triangles = 3n_v + 3n_t$ units



separate triangles:

#	vertex 0	vertex 1	vertex 2
0	(a_x, a_y, a_z)	(b_x, b_y, b_z)	(c_x, c_y, c_z)
1	(b_x, b_y, b_z)	(d_x, d_y, d_z)	(c_x, c_y, c_z)
2	(a_x, a_y, a_z)	(d_x, d_y, d_z)	(b_x, b_y, b_z)

shared vertices:

triangles

#	vertices
0	(0, 1, 2)
1	(1, 3, 2)
2	(0, 3, 1)

vertices

#	position
0	(a_x, a_y, a_z)
1	(b_x, b_y, b_z)
2	(c_x, c_y, c_z)
3	(d_x, d_y, d_z)

Figure 12.6. A three-triangle mesh with four vertices, represented with separate triangles (left) and with shared vertices (right).

- Section 12.1.4 is left to the curious students who want to optimize their code.

Section 12.2 Scene Graphs

- In larger scenes with many objects, it can become a pain to manage how different transformation matrices are applied.
- Most scenes have some hierarchical organization, and using this, we can apply transformations using a scene graph.
- As an example of how scene graphs can be used, consider the following hinged pendulum:

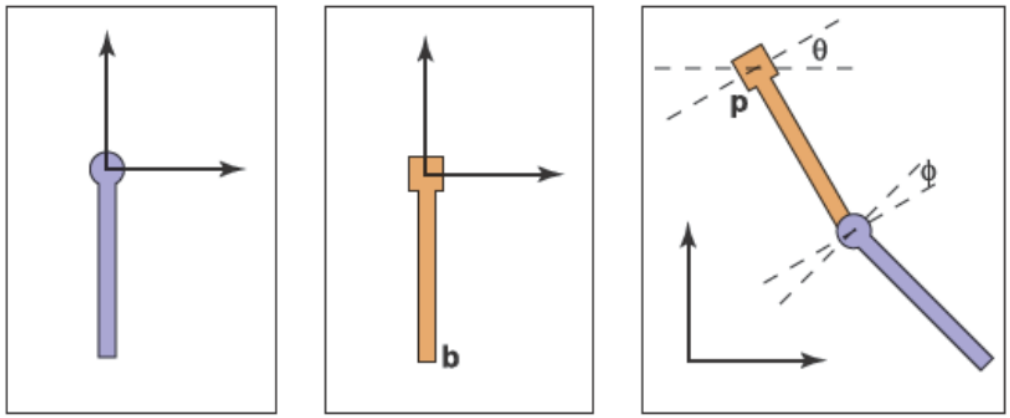


Figure 12.19. A hinged pendulum. On the left are the two pieces in their "local" coordinate systems. The hinge of the bottom piece is at point **b**, and the attachment for the bottom piece is at its local origin. The degrees of freedom for the assembled object are the angles (θ,ϕ) and the location **p** of the top hinge.

- The top pendulum is easy, as it is not dependant on anything else.
- The bottom pendulum, on the other hand, is a problem. Why?
- Using a scene graph data structure, we can easily keep track of point **b**, such that the lower hinge can be moved to the correct place.
- Figure 12.21 may be a good way to create objects for the practicals.

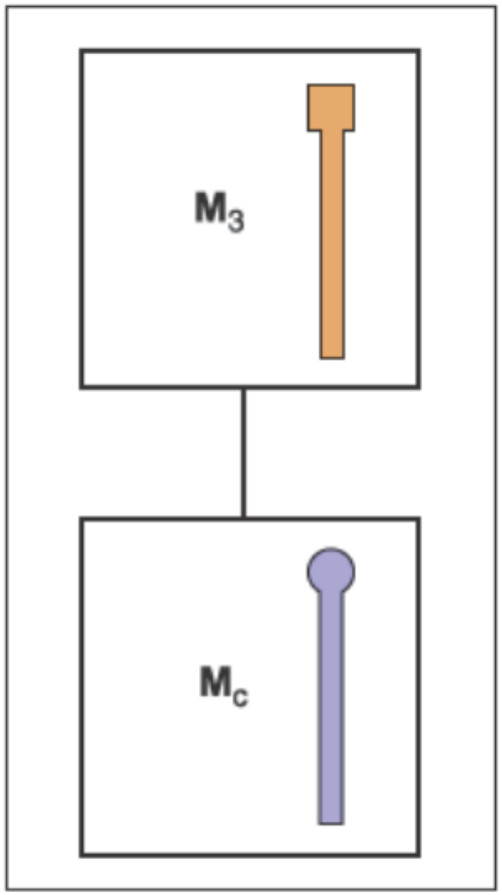


Figure 12.20. The scene graph for the hinged pendulum of Figure 12.19.

The remainder of Chapter 12.2 is left to the curious students.

- Spacial data structures are used to organize objects in a space such that they can be efficiently found.
- Applications?
  - Ray tracing for a light source
- Types of spacial data structures we will quickly discuss:
  - Objects partitioning schemes
    - Structures that group objects together into a hierarchy.
    - Objects are divided into disjoint groups, but groups may overlap in a space.
  - Space partitioning schemes
    - Structures that divide space into disjoint regions.
    - A space is divided into separate partitions, but one object may have to intersect more than one partition.
  - Other
    - Regular: Divide the space into predefined, uniformly shaped boxes.
    - Irregular: The sizes of the boxes are dynamically shaped to accommodate the density of shapes.

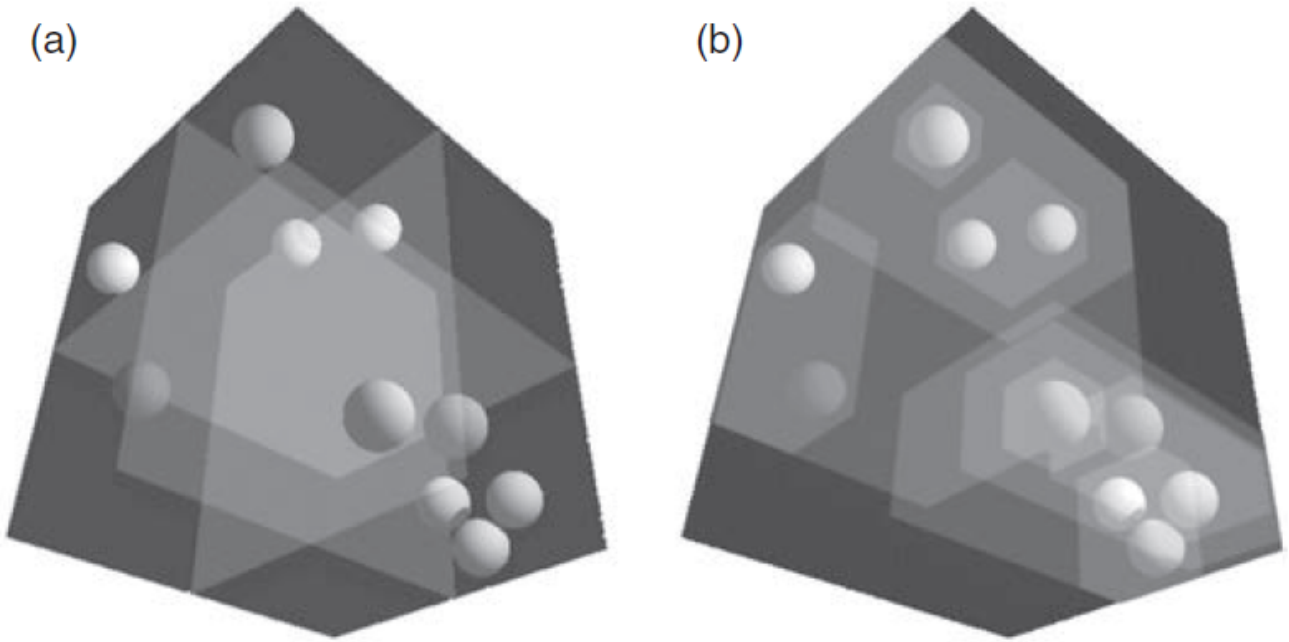
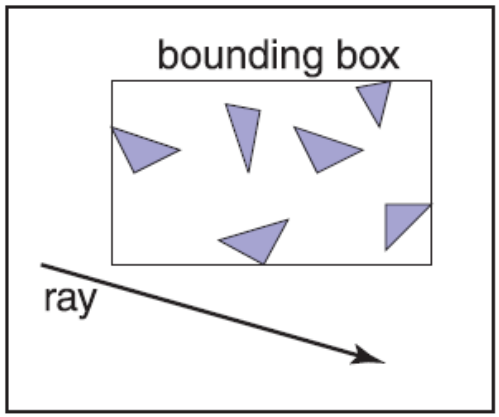


Figure 12.24. (a) A uniform partitioning of space. (b) Adaptive bounding-box hierarchy. *Image courtesy David DeMarle.*

Section 12.3.1: Bounding Boxes

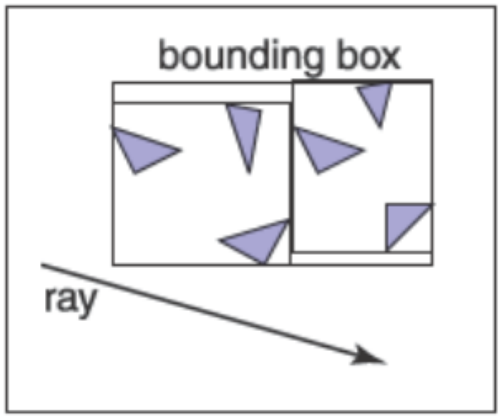
- Place a box around a set of objects and determine if a ray intersects the box or not.
- If it intersects the box, determine which object inside the box is intersected.
- If not, no shape inside the box needs to be considered.
- The remainder of the section describes how to determine if the ray intersects the bounding box.



Section 12.3.2: Hierarchical Bounding Boxes

- Standard bounding boxes have two sides.
  - They are computationally more expensive than the brute force tactic used in earlier chapters, only if the ray intersects the bounding box.
  - They are computationally cheaper than the brute force tactic, when the ray does not intersect the bounding box.
- This can be solved by using a hierarchical bounding box.
  - A bounding box for bounding boxes.

Remainder of the chapter subsection is left to curious students.



Section 12.3.3: Uniform Spatial Subdivision

- Idea is to divide the space into axis aligned cells.
- Each cell points to either a surface that is found in that cell, or null.
- The ray this is traced along its path in this multidimensional (2D/3D) array, until it goes into a cell that contains a surface.
- If the ray then intersects the surface, the ray stops(*), else it continues.
- Other previously discussed strategies can be used for cells that contain multiple surfaces.

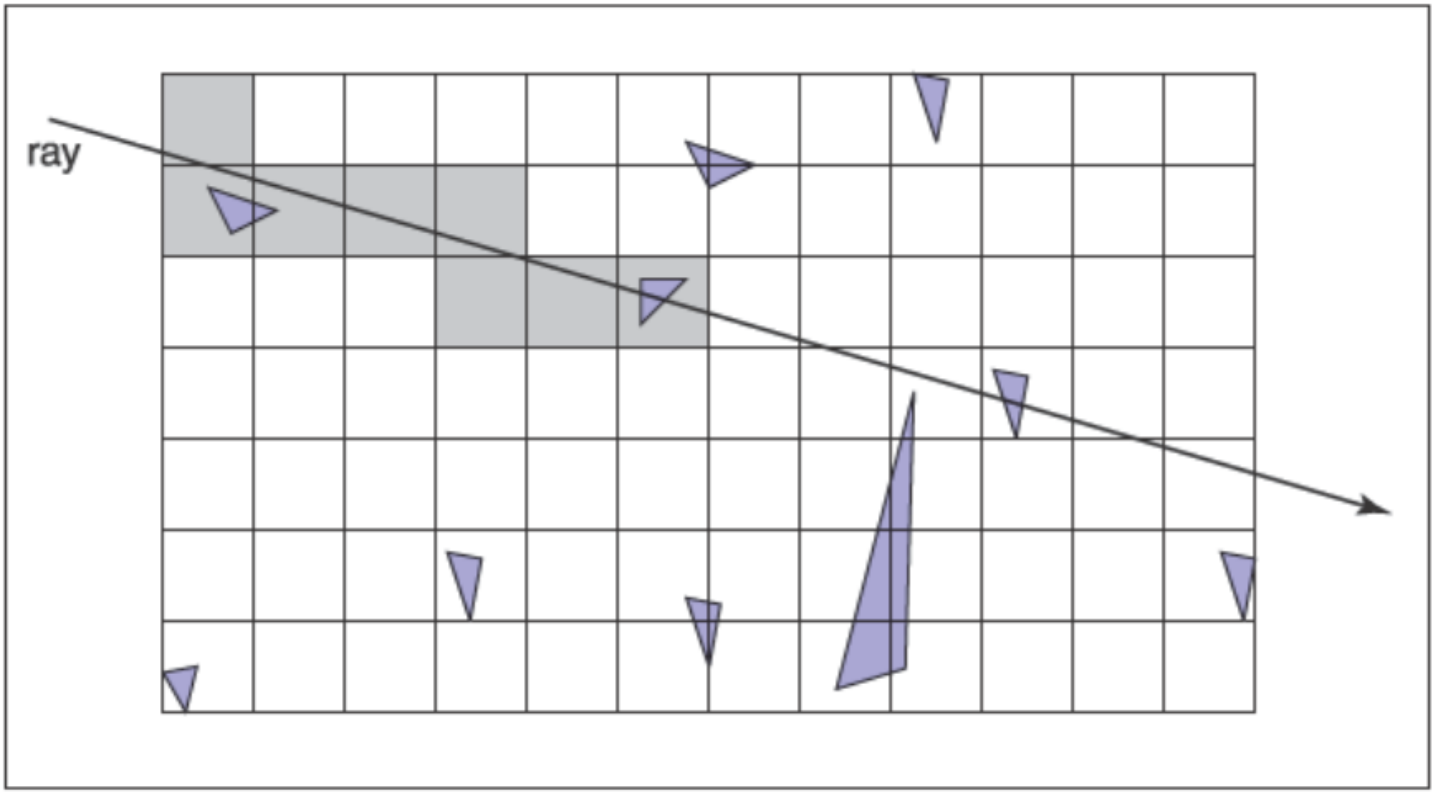


Figure 12.30. In uniform spatial subdivision, the ray is tracked forward through cells until an object in one of those cells is hit. In this example, only objects in the shaded cells are checked.

Remainder of chapter

- Section 12.3.4, Section 12.4, and Section 12.5 are skipped.
- Section 12.4 is an in depth discussion on the Binary Space Partitioning Tree algorithm.
- This can be implemented for bonus marks for Practical 4.
- Section 12.5 uses Tiling MD Arrays which exploit the idea of memory locality to achieve a speedup.

Class Test

- Class test will open at 13h30 and will close at 18h30.
- The test consists out of 4 questions, with a high focus on calculations.
- Round each number to 2 decimal places, else you loose 20% of your mark!
- If you have a question about the test, please make a ticket on Discord.
- You are not allowed to discuss the test with anyone during the 5 hour period.
- You are allowed to use online resources, notes and the textbook to answer and calculate your answers.
- You will only be able to see your final mark for the attempt.
- The system will use the best mark.

Joke of the day - By ChatGPT

Why did the triangle mesh get into a fight with the polygon?

Joke of the day - By ChatGPT

Why did the triangle mesh get into a fight with the polygon?

Because it felt "edgy"!