CS5182 Computer Graphics
Real-Time Rendering

2024/25 Semester A

City University of Hong Kong (DG)

Time-Critical Rendering

- In real-time 3D applications, such as virtual environments and computer games, a very important factor is interactivity.
  - When we perform an action, we expect an instant response from the system, in a way similar to real life interactions.
- In order to achieve real-time feedback, we need to complete the rendering process within a given time, called the frame time (the time between two consecutive image frames).
  - If we generate images at 10 frames per second, a frame time is 0.1s. (Movies have 24 frames per second.)

Time-Critical Rendering

- In time-critical rendering, we trade image accuracy for speed. In other words, given that we have one frame time, we would like to render an image as accurate as possible.

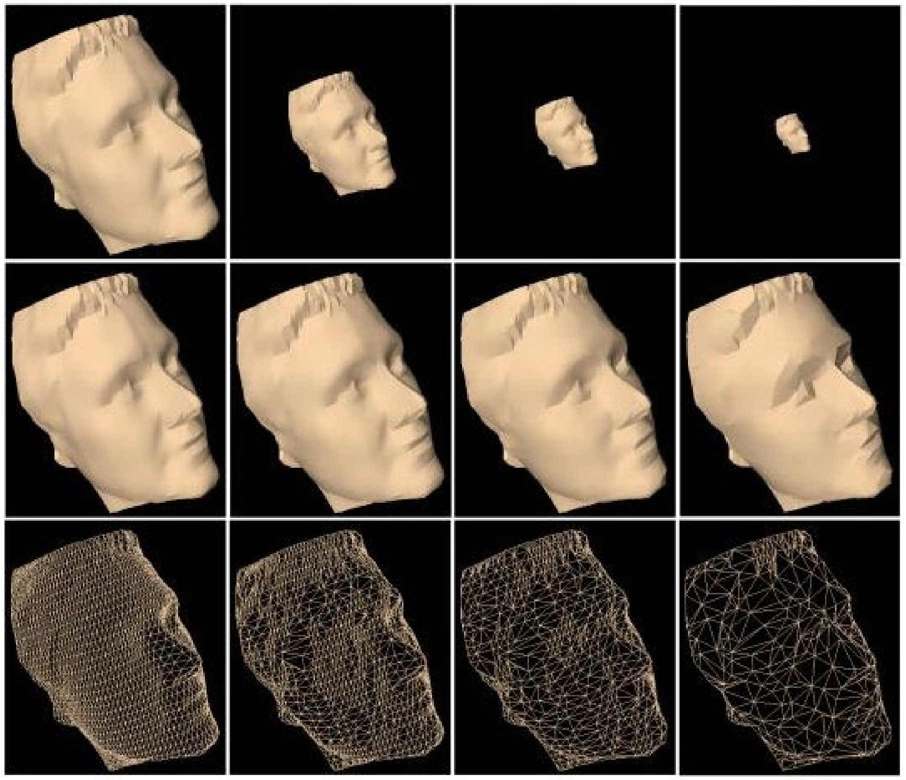
- To achieve this, we need the following:
  - A progressive rendering technique.
  - A way to estimate the visual quality of objects.
  - A way to estimate the rendering costs of objects.

Progressive Rendering

- In general, rendering time is roughly proportional to the number of primitives (usually triangles) to render. To reduce rendering time, we may reduce the number of triangles to represent an object.
- In general, if an object is far away from the viewer, its details may not be clearly visible to the viewer.
- One technique is to produce multiple levels of detail (LoDs)
  - Discrete LoD
  - Progressive mesh

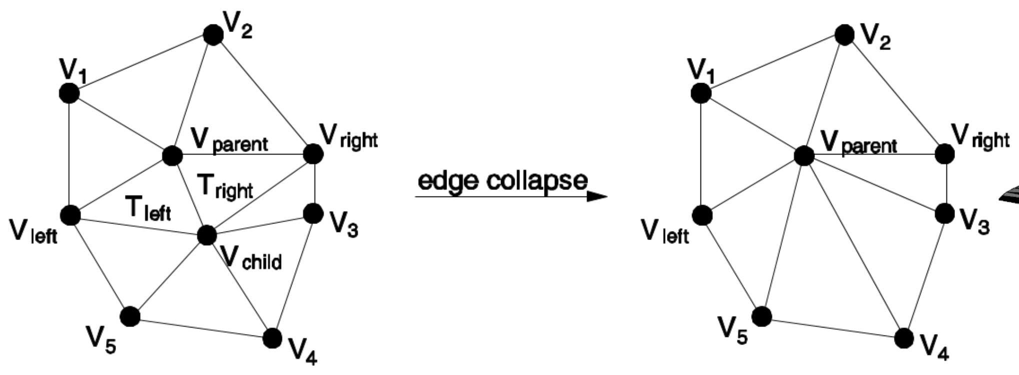
Progressive Rendering

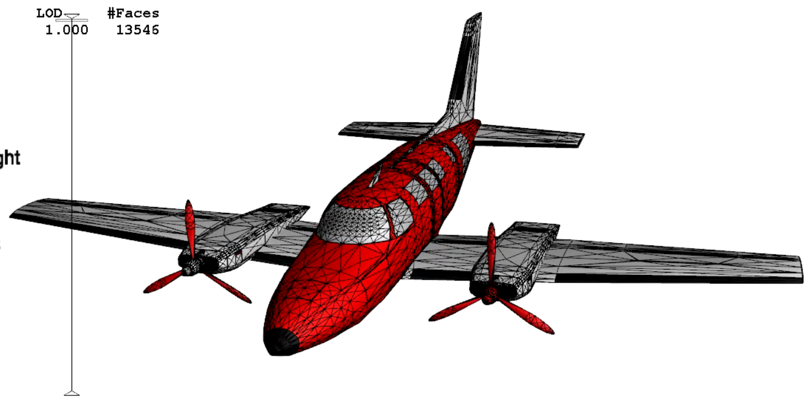
- Discrete LoDs
  - Pre-compute a few models at different resolutions, i.e., different levels of detail (LoDs), for each object.
  - Advantage:
    -- efficient, simply select the most suitable one for rendering
  - Disadvantage:
    -- How many LoDs do we need?
    -- Visual artifacts when switching models.



Progressive Rendering

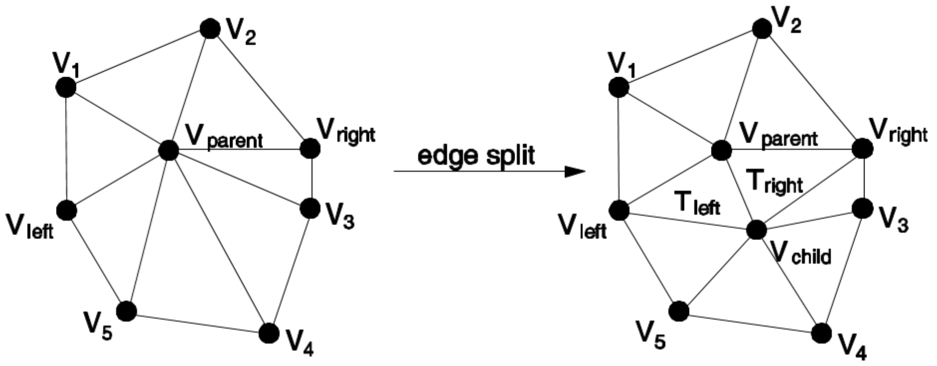
- Progressive meshes
  - The resolution of an object model can be reduced by an operation called *edge collapse*.
  - Each edge collapse operation removes two triangles from the model, i.e., reducing the resolution of the model by a small fraction.
  - If we apply edge collapse to a model continuously, the resolution of the model will be gradually reduced, until it reaches the lowest resolution. This model of lowest resolution is called a base mesh.

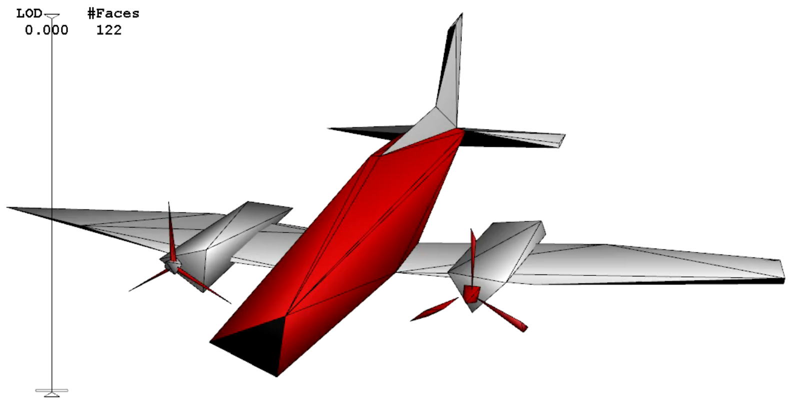




Progressive Rendering

- Progressive meshes
  - The resolution of the model can be gradually increased by a reversed operation called *vertex split*, which introduces two triangles into the model.
  - If we start from the base mesh and we continuously apply edge split to the model, eventually we will reach the highest resolution. The model becomes the original model.



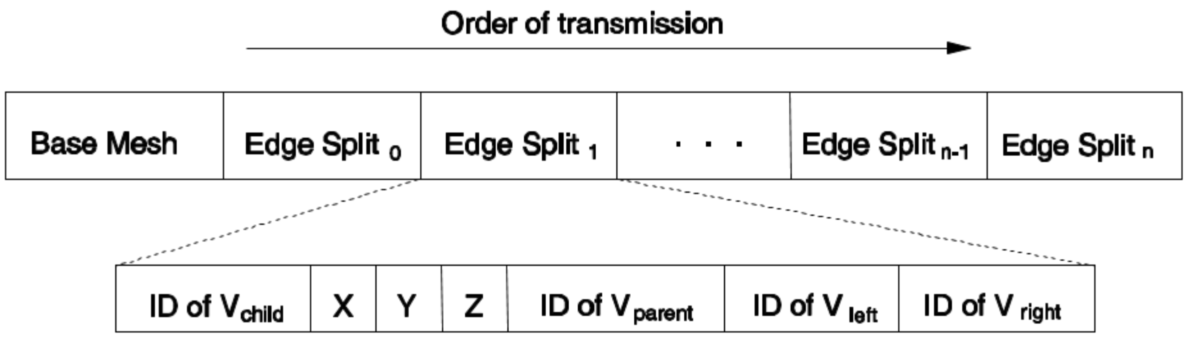


Progressive Rendering

- Progressive meshes
  - The two operations can be represented as follows:

    -- Edge collapse: $(M^{ori} = M^{n}) \xrightarrow{ecol_{n-1}} \ldots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0$

    -- Vertex split: $M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \ldots \xrightarrow{vsplit_{n-1}} (M^n = M^{ori})$
  - Now, we may format the model in the following way:

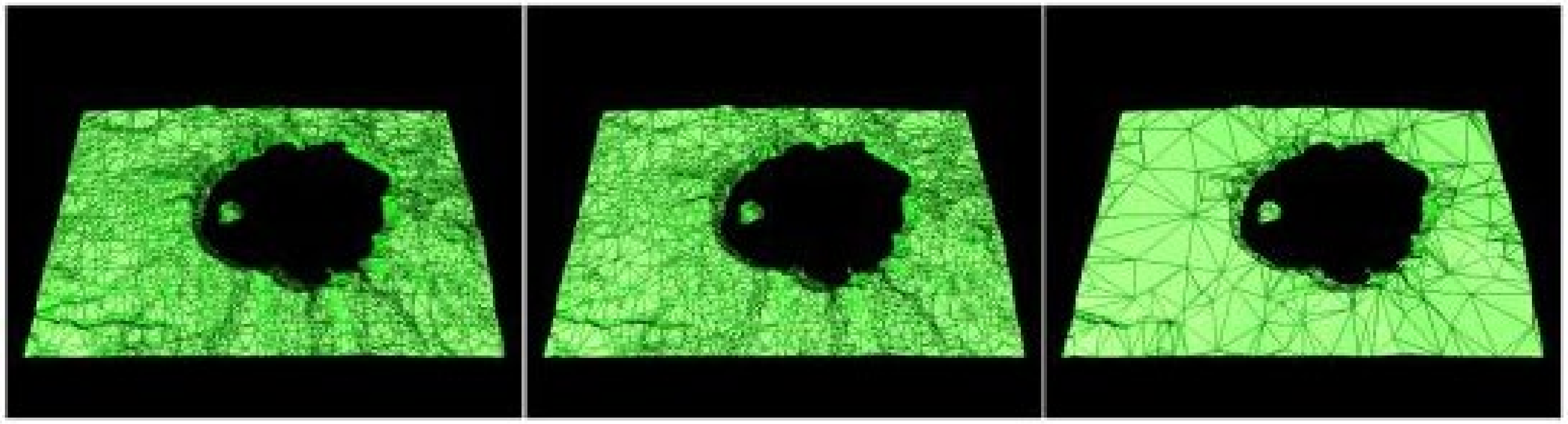


Progressive mesh structure can be used for efficient storage as well as for progressive transmission.

  - The progressive mesh method is very efficient but may not be so effective for large models (to be explained next).

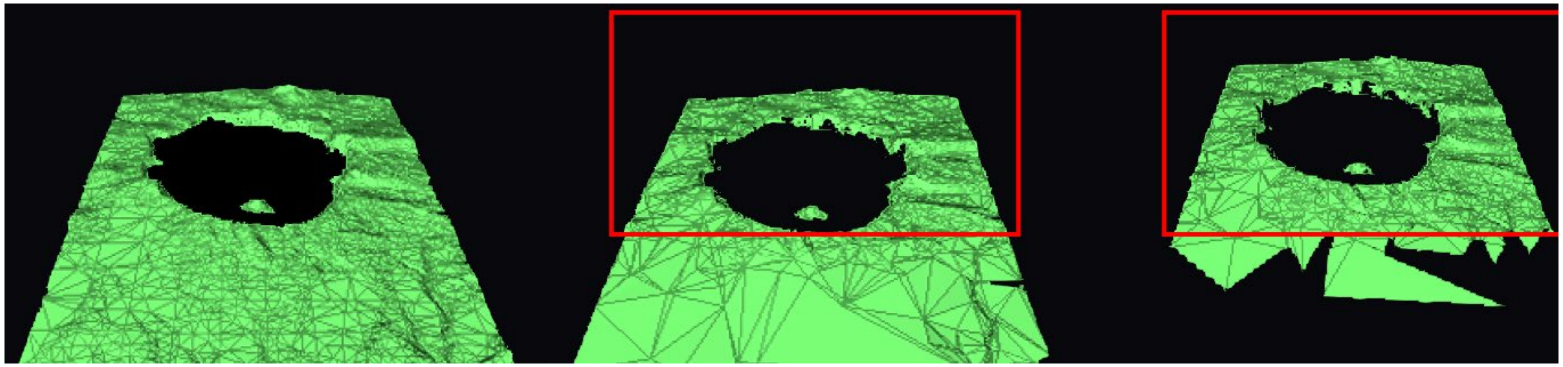
Progressive Rendering

- Selective refinement
  - For a large model, e.g., a landscape model, most of the time we may just be looking at a small part of it in detail.
  - However, a progressive mesh is constructed in a view-independent way, i.e., when in high (resp. low) resolution, the whole model is in high (resp. low) resolution.



Progressive Rendering

- Selective refinement
  - We can optimize the number of triangles, and therefore the rendering performance, if we can selectively refine the model resolution of a local region that we are interested at while keeping the rest at low resolution.



  - The difficulty with selective refinement is that there is a strong dependency between neighboring regions of triangles when we perform edge collapses or vertex splits.

Visual Quality Estimation

- The visual quality is usually measured based on:
  - The model’s level of detail (or number of primitives) used for rendering.
  - The distance of the object from the viewer.
  - The projected size of the object on the screen.

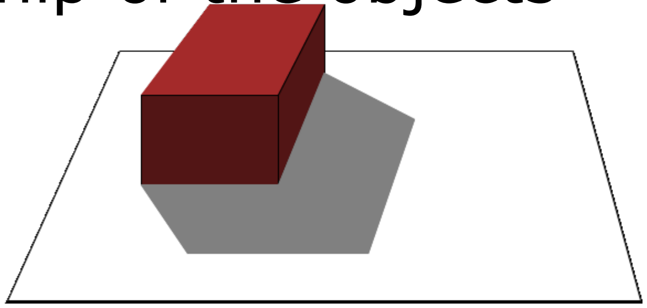
- Other factors that may affect the quality perception of an object:
  - The moving speed of the object.
  - The angular distance of the object from the viewer’s line of sight.

Rendering Cost Estimation

- During rendering time, we need to select the appropriate level of detail and rendering method for each object efficiently so that the maximum visual quality of the output image will be achieved without exceeding the allowable rendering time.
- Therefore, we need a method to estimate the cost of rendering (in terms of rendering time) each object:
  - At whatever level of detail, and
  - Using whatever rendering method, e.g., flat or Gouraud shading.
- This estimation of rendering cost should be done efficiently.

Shadows

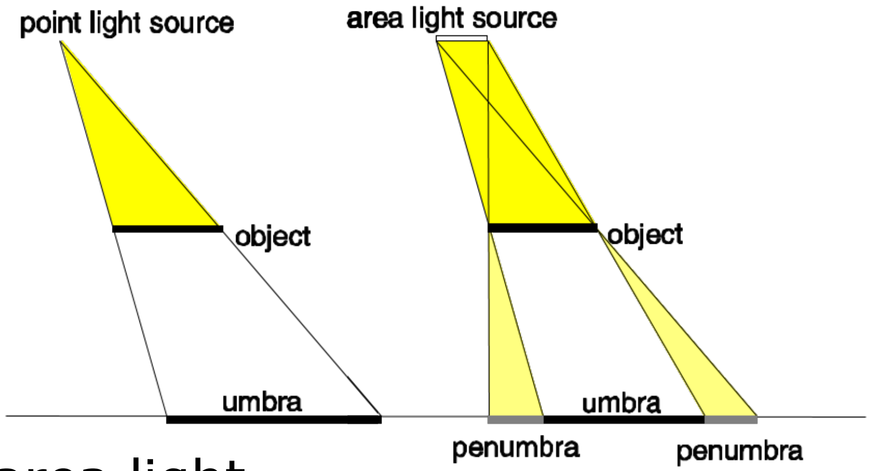
- Shadows are important as they provide information regarding the geometric relationship of the objects and the light sources.



  - Hard shadows

  -- Contain only umbra regions.

  The use of point light sources causes the shadows to have sharp boundaries – resulting in hard shadows.



  - Soft shadows

  -- Contain umbra regions and penumbra regions. The use of area light sources adds soft boundaries to shadows. These soft boundary regions are the penumbra regions.

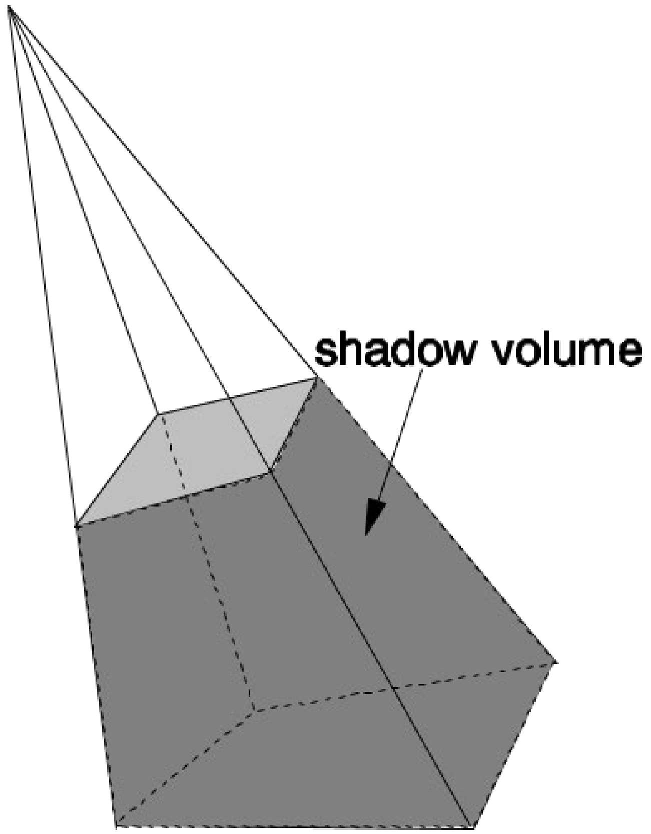
Shadows

- Generation of hard shadows
  - Shadow volume methods

    -- Determine the volumes that are in shadow and bound them with polygons.

    -- Determine if a point is in shadow can be done by checking if this point is inside a shadow volume.

    -- In general, methods of this approach are more complex because they need to handle additional geometry.



Shadows

- Generation of hard shadows
  - Shadow volume methods: determine the shadowing status

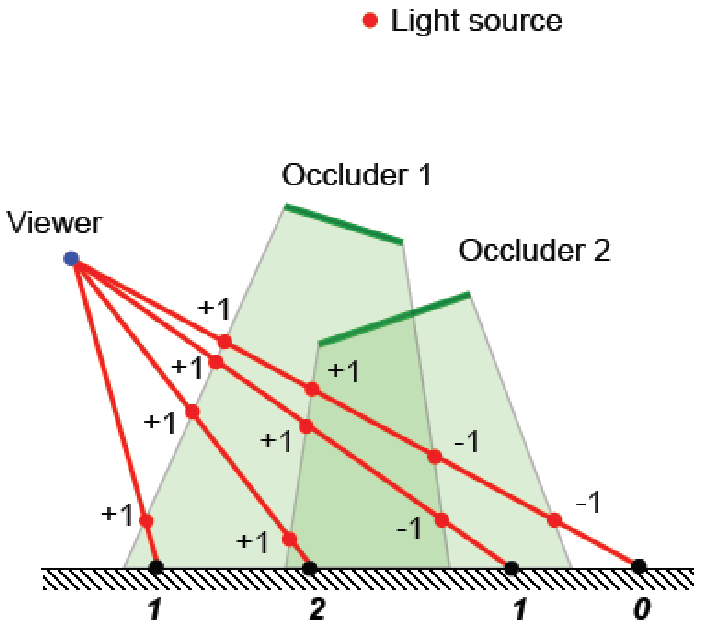
    -- From the view point, we join a line to each point of interest.

    -- We start a counter from zero.

    -- Whenever the line crosses a front-facing polygon of a shadow volume, we add 1 to the counter.

    -- Whenever the line crosses a back-facing polygon of a shadow volume, we subtract 1 from it.

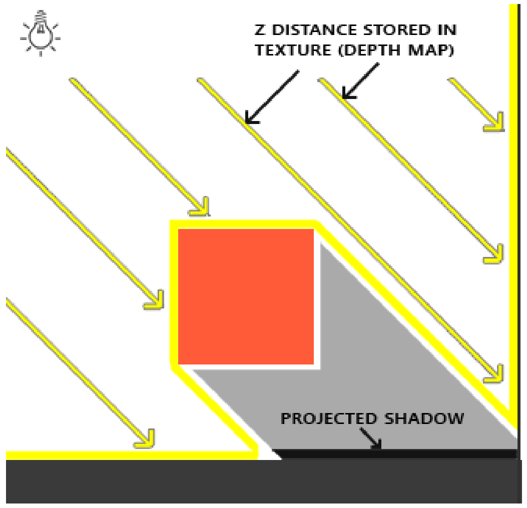
    -- A point is in shadow if the final counter value is a positive number.



Shadows

- Generation of hard shadows
  - Shadow map methods

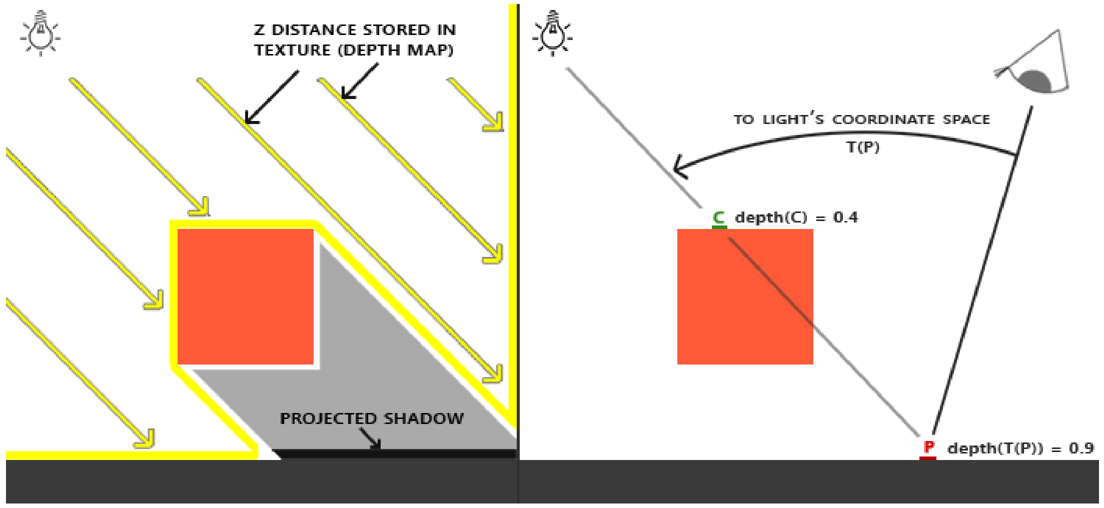
    -- A depth map, called a shadow map, is first rendered (using the z-buffer method) from the point of view of the light source. It indicates objects that are visible to the light source at each pixel of the depth map.



Shadows

- Generation of hard shadows
  - Shadow map methods

    -- Then the output image, from the viewer's point of view, is generated. For each pixel computed, we transform the depth value at the viewer's space to the light source space.

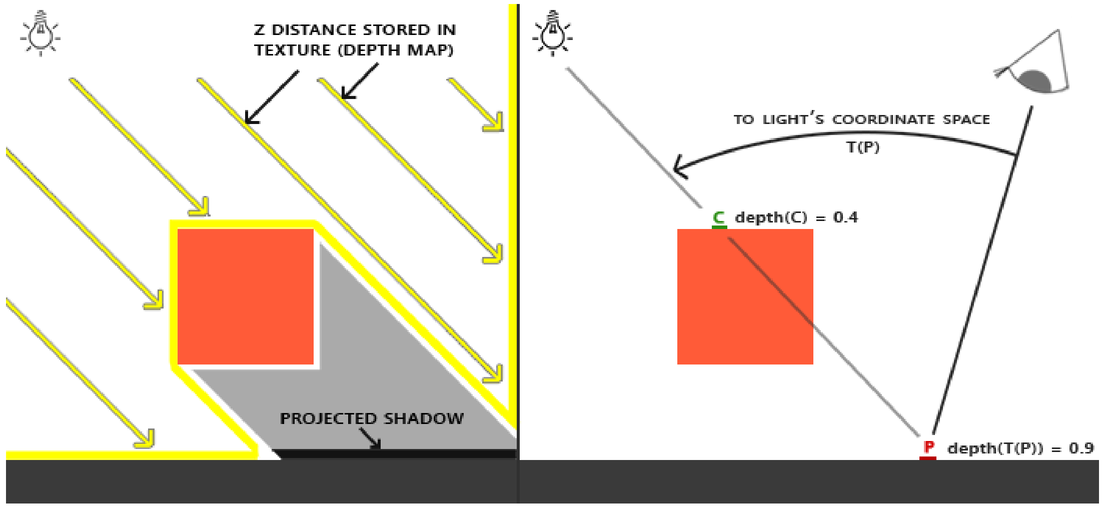


Z DISTANCE STORED IN TEXTURE (DEPTH MAP)
PROJECTED SHADOW
TO LIGHT'S COORDINATE SPACE
T(P)
C depth(C) = 0.4
P depth(T(P)) = 0.9

Shadows

- Generation of hard shadows
  - Shadow map methods

    -- We compare the transformed depth value with the corresponding depth value in the shadow map. The point is in shadow if the transformed depth value is larger than the depth value in the shadow map.

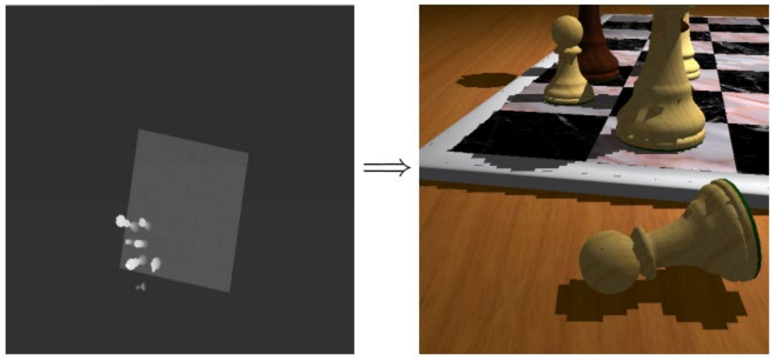

Z DISTANCE STORED IN TEXTURE (DEPTH MAP)
PROJECTED SHADOW
TO LIGHT'S COORDINATE SPACE
T(P)
C depth(C) = 0.4
P depth(T(P)) = 0.9

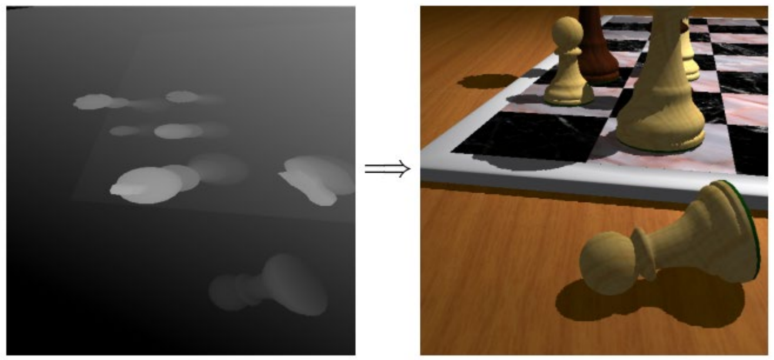
Shadows

- Generation of hard shadows
  - Shadow map methods

    -- The major limitation of the standard shadow map method is that it suffers from aliasing.

    -- To address this problem, the *perspective shadow map* method suggests to first transform all objects perspectively from the point of view of the viewer before generating the shadow map. Hence, objects near to the viewer will appear larger in the shadow map while objects further away will appear smaller in the shadow map.


shadow map and resulting image


perspective shadow map and resulting image

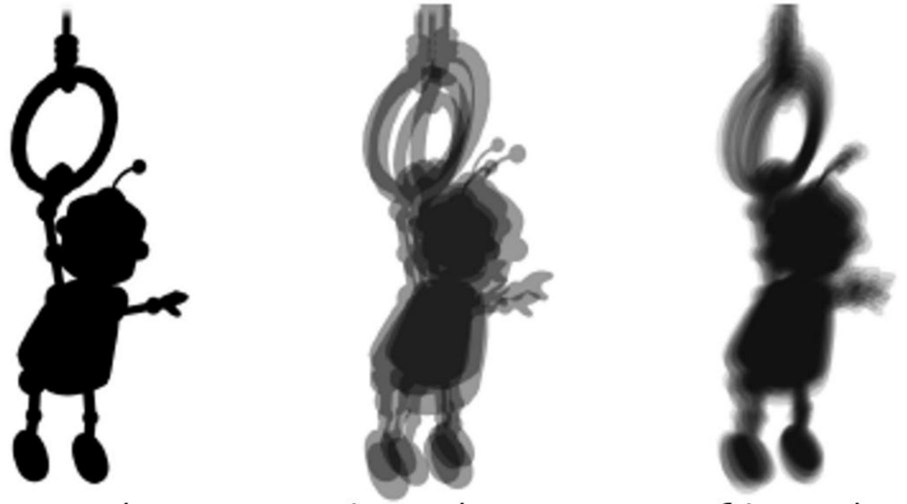
Shadows

- Generation of soft shadows
  - Combining point light sources

    -- Sample the light source multiple times at different locations and produce a shadow map for each light source sample point.

    -- These shadow maps are combined to form an attenuation map.

    -- This attenuation map can then be used to modulate the illumination of each shadowed point during the final image rendering stage.


one sample 4 samples 64 samples

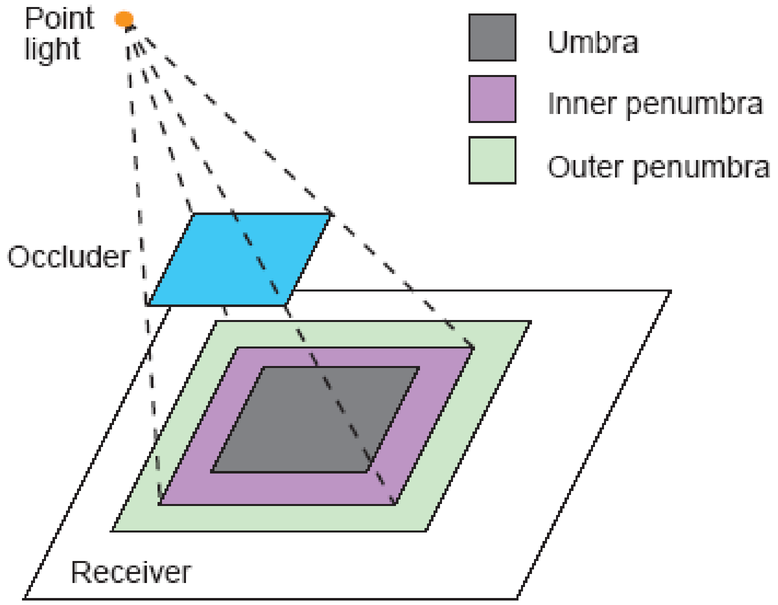
Shadows

- Generation of soft shadows
  - Single sample soft shadow

    -- A standard shadow map is first generated from the light source.

    -- Based on the depth of the occluder, a region is created outside the shadow boundary to form a penumbra region.

    -- Likewise, another region is created outside the shadow boundary to form another penumbra region.

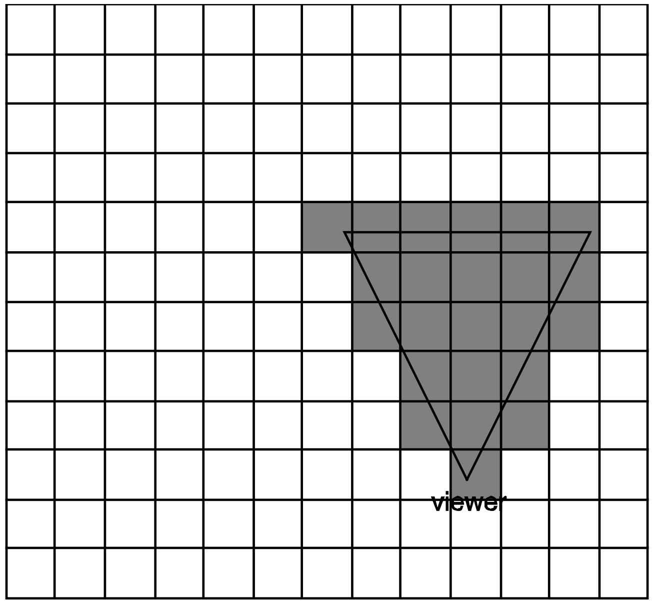


-- An attenuation factor is then produced from the inner edge of the inner penumbra region (of value 1) to the outer edge of the outer penumbra region (of value 0).

Visible Object Determination

- In a large scene, containing a large number of objects, it may be too inefficient to process all the objects in the scene in order to render a single image. Therefore, we need a way to quickly identify the potentially visible objects.
  - One way to do it is to divide the scene into small cells.

All cells overlapped with the view region are determined. Only objects intersecting the overlapped cells are considered for rendering.

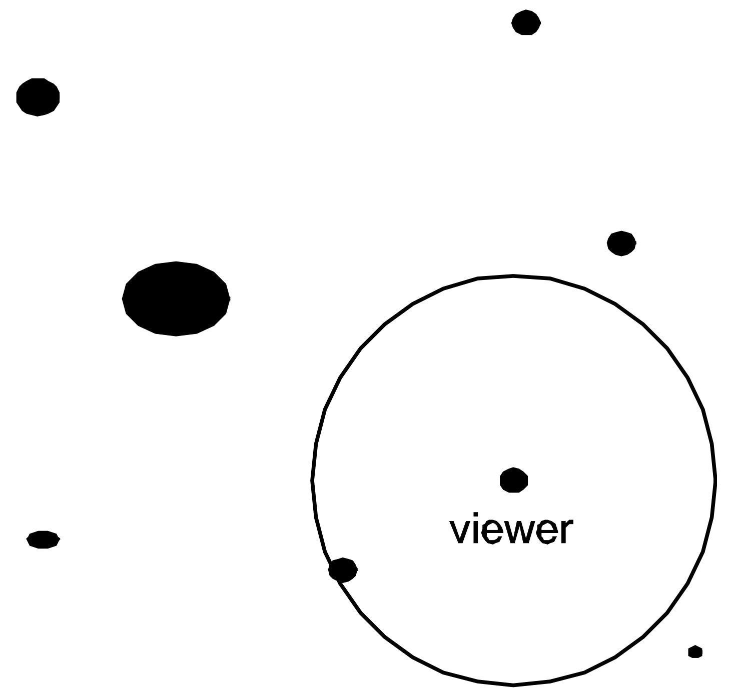


Visible Object Determination

- Another approach is called the Area of Interest (AOI).

    -- Only objects which are inside the viewer’s AOI are considered as visible.

    -- Visibility can be determined by comparing the Euclidean distance between the viewer and the object center with the radius of the AOI.

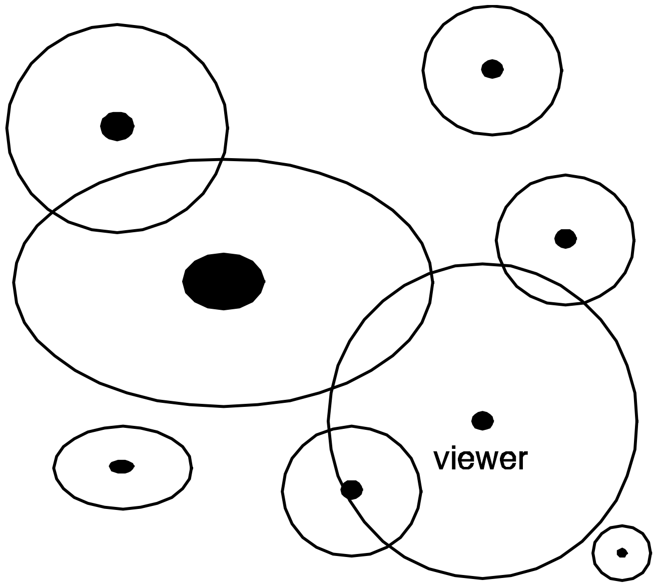


Visible Object Determination

- We can generalize the AOI approach to cover the objects too

  -- Only objects whose AOIs overlap with the viewer's AOI are considered as visible to the viewer.

  -- Visibility can be determined by comparing the Euclidean distance between the viewer and the object center with the sum of the two radii.



Visible Object Determination

- We can further extend the AOI concept by subdividing the scene into small cells.

-- Each cell maintains a list indicating all objects whose AOIs overlap the cell.

-- Visible objects can now be determined very efficiently by checking the list in each cell covered by the AOI of the viewer.