Foundations of rendering

Cornerstones of image synthesis

- 3D scene
- **Rendering**
- Raster image

Recap

- In previous lectures we have discussed how to model elements of 3D scene:
  - 3D model
  - Light
  - Camera
- Now, we will discuss how to combine scene elements to simulate interaction of light to create an image

Rendering: intuition

- Photography: Taking a photo from a real world:
  - Light source is emitting light into the space
  - Light travels through space and interacts with objects
  - Small portion of that light falls into camera, where image is created
- Physics: Interaction of light with objects in space and camera is well described in physics (wave and geometrical optics).
- Rendering is computer graphics tool which simulates light transport and interaction of light with objects in order to create an image.
  - It creates viewable 2D image from 3D scene

Rendering tasks

- In order to create an image from 3D scene, rendering algorithm must answer:
  - Visibility (perspective/orthographic projection)
  - Light transport
  - Shading

Visibility: high level

- Calcualte which objects are visible from camera
- Utilize perspective/orthographic projection

Light transport: high level

- For each object that is visible, compute incoming light
- This method also relies on concept of visibility but between surfaces and lights in 3D scene

Shading: high level

- Use intersected point shape and material information combined with incoming light to calculate color and intensity of that point looking from camera.

Practical rendering approaches

Two main practical rendering approaches are:

- Ray-tracing based rendering
- Rasterization based rendering

Raytracing and rasterization

- Raytracing is historically early method for image synthesis based on 3D scene description
- Raytracing has always been recognized as best method for generating photo-realistic images. However, up until 1990-2000s it hasn't been used widespread
- Due to hardware advancements, rasterization-based rendering gained a lot of attention
- Raytracing is intuitive and straightforward method of simulating physical phenomena of how image is created. Based upon raytracing, more advanced rendering methods are build. Furthermore, understanding raytracing gives good foundations for understanding rasterization-based rendering systems.

Note: compexity of scenes and rendering

- Although HW and methods advanced, the time of rendering the scenes stayed the same! This is because complexity of the 3D scenes as well as rendering algorithms increased.
  - **Blinn law**

Rendering: intuition using raytracing

- To produce an image, first, we need information from where we look at 3D scene and a 2D surface on which 3D scene will be projected as an image.
  - In lesson on camera, we mentioned exactly these elements: eye and film plane. Therefore, for these purposes, we can use pinhole camera.
  - Captured image color and brightness of objects in 3D scene is a result of light and objects (shape and material) interaction
- To figure out interaction of light and objects we need information about objects in 3D scene as well as light sources.
  - Without light we wouldn't see objects, without objects we wouldn't see light.
  - Using this information, we can simulate light-object interaction.
- Simulating physical (real-world) light-object interaction means simulating all light rays paths from light and their interactions with objects. Furthermore, only small amount of that light actually falls on camera. This kind of simulation is called **forward ray-tracing or light tracing.** Simulating this is not tractable! Therefore, we simplify:
  - First, Only light-object interaction that are visible from camera should be simulated
  - Second, we reverse the process: we start with rays that fall into camera and build from there.
- Based on previous discussion, we trace rays from sensor to the objects. This simulation is called **backward ray-tracing or eye-tracing***.

* "An Improved Illumination Model for Shaded Display" - Turner Whitted

Backward ray-tracing*

- Based on previous discussion, now we understand the first steps:
  - Generate ray using camera (eye and film) → primary/camera/visibility ray
  - Trace ray into the scene
- Tracing ray into the scene can end up in following possibilities:
  - Intersects with object
  - No intersection →hit background
- In the case where ray hits the object, two main steps are done:
  - Calculate amount of light (incoming) falling on intersection point → shadow/light ray.
    - This ray can be directly traced from intersection to light source → direct illumination
    - Advanced: sample various directions from this point in 3D scene to obtain reflections of other objects → global illumination
  - Calculate amount of light reflected in primary ray direction using incoming light and material specification

* Method in computer graphics using the concept of shooting and following rays from light or eye is called path-tracing.

Practical raytracing* : algorithm

Algorithm:

- For each pixel in raster image:
  - Generate primary ray by connecting eye and film using camera information
  - Shoot primary ray into the scene
  - For each object in the scene
    - Check if intersected with primary ray. If intersect multiple objects, take one closest to the eye
  - For intersection, generate shadow ray from intersection point to all lights in the scene
    - If shadow ray is not intersecting anything, lit the pixel

<IMAGE: RAYTRACING>

* Arthur Appel in 1969 - "Some Techniques for Shading Machine Renderings of Solids"

Three steps of raytracing

- Generate ray for each pixel of film plane → camera ray
- Ray-geometry intersection: testing intersection of generated ray and 3D scene objects to obtain what is visible from camera
- Shading: calculating the color and intensity of intersected points
  - Requires light-matter calculation
  - Requires light transport for gathering incoming light

Raytracing vs rasterization

- As we can see from the algorithm, **raytracing** iterates through all pixels of image plane, generates rays and then iterates over all objects in the scene. This approach is called **image centric.**
- On the other hand, **rasterization** is reversed: it loops over all geometric primitives in the scene (triangles), projects these primitives on image plane and then loops over all pixels of the image plane. This approach is called **object centric**.

Rasterization: quick peek algorithm

- TODO

Rasterization and raytracing

- Based on current discussion, we can conclude that both **raytracing and rasterization are used to solve the visibility problem** (which also contains **perspective/orhographic projection** solving).
- Solving the visibility problem is only a part of rendering. Other part is determining the color and intensity of visible surface. This step is called **shading** and it uses **light transport** to gather light on visible surface and calculates light-matter interaction

Practical raytracing: notes

- Raytracing method can be separated in:
  - **Determining visibility from camera**: determine which point in 3D scene is visible for each pixel of image. Note that **perspective projection** is inherent due to camera which is used for generating rays
  - **Shading**: calculating the color of visible point – this operation besides light-matter calculation also requires **light transport** to collect all light which influences the color and intensity of that point.
- Both steps require extremely time-consuming intersections between rays and scene objects (geometry)
  - Efficient ray-object intersection method is needed
  - 3D scene must be represented for fast and efficient spatial search of objects which should be tested for intesection
- On the other hand, rasterization can solve the visibility problem (camera-object) very fast. That is why real-time graphics is predominantly using GPU rasterization approach. But when it comes to shading, due to the available information on 3D scene, it is not as good as raytracing.
- Raytracing is slow for solving visibility problem, but it enables high-quality shading. For high-quality, photo-realistic production rendering, ray-tracing is almost always used*.

* However, real-time ray-tracing is now possible to certain degree with certain hardware and it is hot research topic!

Extending basic raytracing

- https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/adding-reflection-and-refraction
-

Importance of visibility calculation for rendering

- Based on previous discussions, we can highlight the importance of visibility calculation and how it is related to all concepts in rendering.
- Visibility problem is concerned with determining visibility between points
- First, when determining visible objects from camera, we solve visibility problem and utilize perspective/orthographic projection
- Then, when calculating shading we require light transport information which is again based on visibility between surface and lights in 3D scene
  - This is required to calculate shadows, soft shadows, global illumination effects such as reflection, refraction, indirect reflection and.
- Rasterization is good for solving the visibility from camera to object. But it is not good for finding visibility between surfaces in 3D scene which is important for light transport and shading
- Raytracing is good for solving the visibility from camera to object. And it also can be used for finding visibility between surfaces in 3D scene which is important for light transport and shading

<IMAGE: visibility in rasterization and raytracing>

Rasterization- vs Raytracing-based production rendering software

- Examples of rasterization-based renderers in production
  - Godot, Unity, Unreal
- Examples of raytracing-based renderers in production
  - Appleseed, Arnold, etc.

<IMAGE: differences and comparison between rasterizers and raytracers>