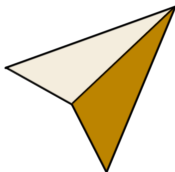


COMP4610/COMP6461

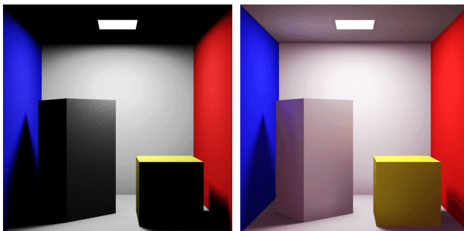
Week 11 - Radiosity, CSG, and Blender

<Print version>

Radiosity and other Photo-realistic Methods

Radiosity

- Radiosity is all about tracking how energy radiates (via photons) from one surface to another (works with heat too!).
- This involves implementing physical formulations of flux (radiant power).
- We only calculate viewer independent lighting, which means no specular light, only diffuse.
- An exact solution can be found by solving a (large) set of simultaneous equations. However, usually an iterative approach is used instead.



Left direct lighting only **Right** indirect lighting via radiosity.
https://en.wikipedia.org/wiki/Radiosity_(computer_graphics)

Radiant-Energy

- The energy of one photon (in joules) is: $E = hf$, where $h = 6.6 \times 10^{-34}$, and f is the frequency.
- Sum them over all frequencies. This is known as spectral radiance: $E = \sum_f \sum_{photos} hf$
- The radiant flux, or radiant power is (joules/sec or watts): $\Phi = \frac{dE}{dt}$
- The radiosity is the radient flux per unit area leaving a surface (watts/meter^2): $B = \frac{d\Phi}{dA}$
- Intensity (or radiance) is radiant flux in a particular direction per unit solid angle per unit area (watt/(m^2·steradians)). $L = \frac{d^2\Phi}{dA\, d\Omega \cos\theta}$

Basic Radiosity Model

- Lighting is calculated in a closed system in a viewer independent way.
- Assume all surfaces are: small, opaque, and ideal diffuse reflectors.
- The scene can be broken up into may small patches.
- Given n patches the radiant energy from patch k will be:

$$B_k = E_k + \rho_k \sum_{j=1}^{n} B_j F_{jk} \frac{A_j}{A_k}$$

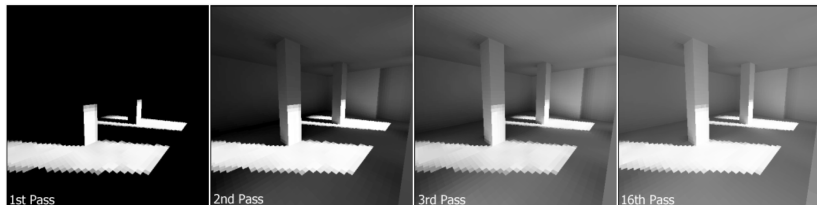
- Emitted radiant energy.
- Percent of incident light that is reflected in all directions.
- Form factor (fraction of energy leaving patch j and arriving at patch k)
- This can be written in matrix form and solved, but the system of linear equations is very large.

Progressive Refinement

Progressive refinement uses the following approach:

- The radiosity values B_i are initialised to the emitter values E_i (mostly zero).
- Then repeatedly update patches where B_j is set to the radiance from the previous pass.
- This can be repeated until the change in radiosity is within our tolerance, or for a fixed number of passes.

Progressive Refinement

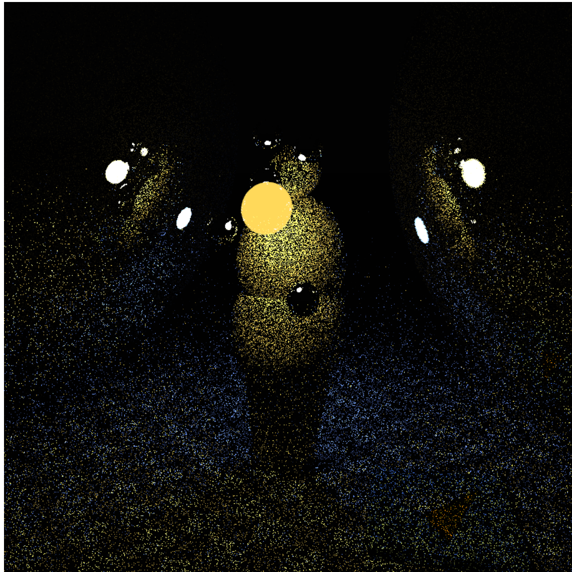


Credit https://en.wikipedia.org/wiki/Radiosity_(computer_graphics)

Path Tracing

- **Global illumination** effects can be created by making use of **Monte Carlo** approaches.
- Path tracing is like ray tracing with rays shot through pixels of the screen, however, rather than just use perfectly reflected/refracted angles, the ray is randomly scattered based on the **BRDF**. As the ray is bounced around the scene colour from direct light is incorporated. Many rays are shot through the same pixel and the result is averaged.
- Path tracing will simulate effects such as soft shadows, caustics, ambient occlusion, and indirect light.
- If the scene is accurately modelled then path tracing will produce photo realistic images. However, there is an enormous computation cost.

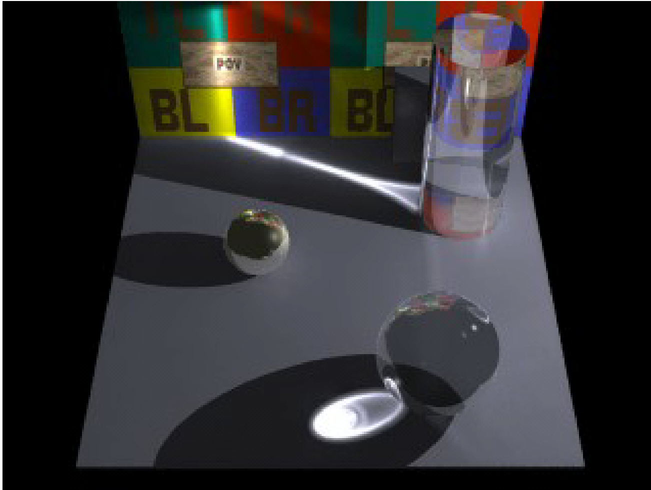
Path-tracing (16 samples)



Photon Mapping

- Photon mapping is a **global illumination** approach that first creates a photon map. This is created by randomly shooting photons from the light sources into the scene. When photons hit the surfaces of the scene they are added to the photon map, they are also recursively reflect from the surfaces based on the **BRDF**.
- The Photon map is stored separated from the scene structure in a BSP-tree (such as a kd-tree).
- In the second stage of photon mapping a ray tracing approach is use where the photon map provides the lighting information for the surfaces struck by rays.
- Like path tracing photon mapping is also a Monte Carlo approach, however, it is not as computationally demanding to produce a similar result.

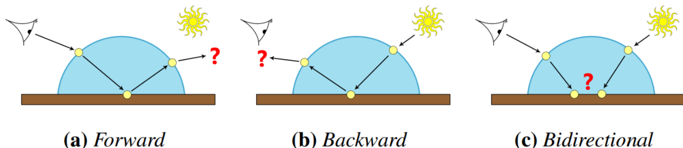
Photon Mapping Example



Photon mapping example from the POV-Ray raytracer.

Bidirectional Path Tracing

- It's much more efficient to trace from the camera to the light than the other way around (as most photons do not end up hitting the camera lense).
- However some lighting effects work best (or require) that we trace from the light to the camera (caustics).
- Bidirectional tracing aims to address this by tracing in both directions.
- https://graphics.pixar.com/library/CausticConnections/paper.pdf



The Rendering Equation

Introduced in 1986 [1], the rendering equation describes the equilibrium radians leaving a point as an integral

$$L_{\mathsf{o}}(\mathbf{x}, \omega_{\mathsf{o}}, \lambda, t) = L_{\mathsf{e}}(\mathbf{x}, \omega_{\mathsf{o}}, \lambda, t) + \int_{\Omega} f_{\mathsf{r}}(\mathbf{x}, \omega_{\mathsf{i}}, \omega_{\mathsf{o}}, \lambda, t) L_{\mathsf{i}}(\mathbf{x}, \omega_{\mathsf{i}}, \lambda, t)(\omega_{\mathsf{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathsf{i}}$$

This was derived from **conservation of energy**. Most of the lighting models we have covered in this coarse can be considered approximate solutions to this integral.

- L_{e} is the emitted spectral radiance.
- L_{i} is incoming spectral radiance.
- f_r is the BRDF (which we can measure), but now includes the wavelength λ and time t.

Solid Modeling

Solid Modeling

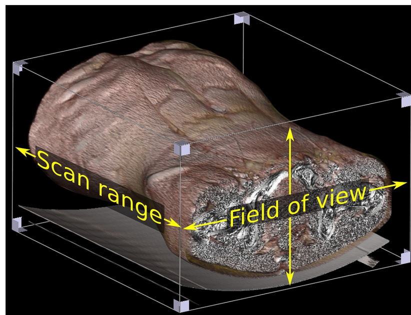
- Most 3D engines only model an objects **surface**.
- This simplification is usually good enough, but does have some limitations, such as what happens if the camera is inside the object, what is the objects physical mass etc?
- Solid modelling attempts to model 3d objects with an emphasis on physical fidelity.
- Useful for CAD / simulations.

Often modeling a scene involves trade-offs between:

- How effectively the approach can model the aspect,
- the amount of processing the approach requires when rendering,
- storage required.

Voxels

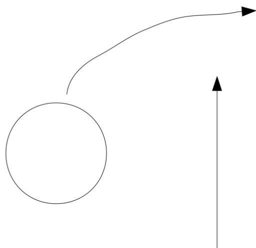
- Perhaps the simplest way to model solid objects is via **voxels**.
- A voxel is a cube of fixed size, arranges in a fixed spatial grid. (the cube is the only space filling platonic solid)
- Used in medicine (CT scans).



Credit https://en.wikipedia.org/wiki/Voxel

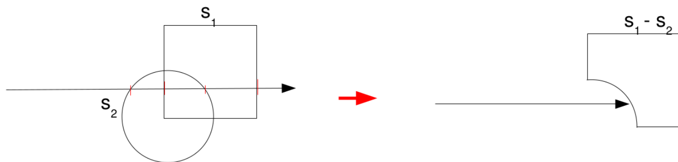
Swept surfaces

- A 3D object can be created by sweeping a shape through space.
- Such representations will often be transformed into polygons to be rendered within the graphics pipeline.



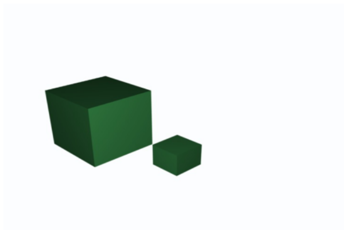
Constructive Solid Geometry

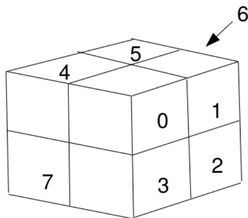
- Primitive solid shapes such as boxes, spheres, cones, etc... can be used in conjunction with union, intersection, and difference operators to construct complex shapes.
- Ray casting can be used directly on this representation for rendering.

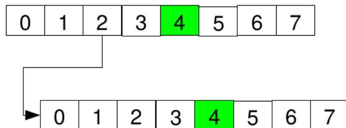


Octrees

An octree recursively divides cubes into eight smaller cubes when a region of space is not uniform. Uniform regions form the leaf nodes of the octree.



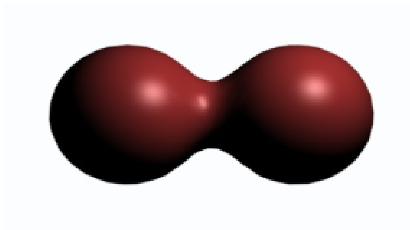




Blobby Objects

Objects can be defined by a sum of distribution functions over a region of space. The surface of such objects is then defined to be the points in space that sum to a particular value.

$$\sum_k b_k e^{a_k r_k^2} = T \tag{1}$$



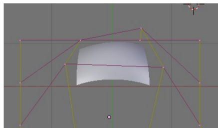
Surface Modeling

Polygons

- A boundary representation that encloses the interior of an object can be used to model most objects. These use a set of surface polygons. Such polygons can be efficiently rendered due to their simple linear form.
- Often other representations will be reduced to a set polygons to be included into the rendering pipeline.
- The visual effectiveness of such representation can be greatly improved incorporating techniques such as:
  - textures,
  - vertex norms, and
  - bump mapping.

NURBS (high level surfaces)

- Often a system will use call-back functions to enable complex surfaces.
- Non-uniformm rational B-splines (NURBS) are a generalization of both Bezier splines and B-splines and provide a very powerful and effective way of representing a surface.
- To construct NURBS you need:
    - control points,
    - the degree of the polynomial,
    - weight factors, and
    - a knot vector.
- OpenGL has NURBS.



Particle Systems

- Particle systems can be used to add visual effects like: smoke, moving water, falling leaves, fog, snow, dust, hair, sparks, 'magic' effects, etc.
- Simple physics that models position, velocity, acceleration, and collision can be used.
- 'Static particles' are rendered over their entire lifecycle. This can produce effects like hair.
- Particles are normally rendered using a quad that are always facing the viewer (billboard quad).

A cube emitting 5000 animated particles, obeying a "gravitational" force in the negative Y direction.

The same cube emitter rendered using static particles, or strands.

Blender

Nanite (geometry virtualization)

Geometry Virtualization

Nanite is the virtual geometry system used in Unreal Engine 5.0.

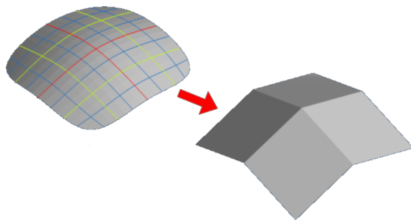
These notes are based on the Siggraph 2021 presentation found here and here.

Nanite

Credit www.unrealengine.com

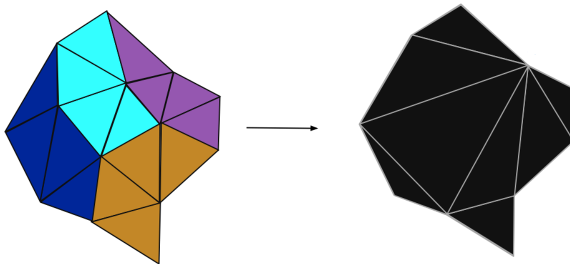
The Problem

- Rescaling textures is straightforward, ‘rescaling’ meshes (in terms of the number of vertices) is very hard.
- Converting to voxels looks bad and is too large storage wise.
- Ideally, we want a method that can take a high resolution mesh and quickly re-sample it (just like mipmaps).
- Once we have this, we can just import film grade meshes into our game and have them ‘just work’.



Mesh Simplify

Works by breaking the mesh into clusters then simplifying the cluster. Note that the boundary remains unchanged.



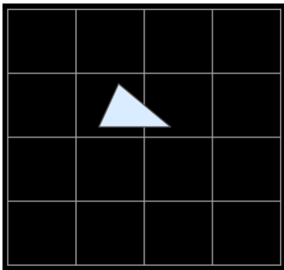
Making this work without cracks is difficult.

Mesh Simplify

- The idea is to (in realtime) simplify meshes by collapsing edges.
- This took at least 1-year full time to do.
- Returns an estimate of the vertex error in worlds space, which is projected onto the screen to get pixel error.
- Because of this, we can simplify a mesh down to the point where errors are sub-pixel. Because of this it would be almost impossible to see any difference.
- Might not work with shiny materials, where small changes in normal make a big difference.
- I'm surprised this worked. Many people have tried, and failed, to get this going

Micropoly Software Rasterizer

- Using this process, many triangles end up being only a few pixels wide.
- Surprisingly, they found that a **software** rasterizer outperformed the **GPU** for these triangles (by 3x).
- Both the **GPU** and **CPU** work on drawing the triangles, CPU for small clusters, GPU for large.



Nanite: What it doesn't do

- Only works with static meshes.
- Not great for many small things (grass, leaves, hair).
- No translucency support.

Next Week

Next week we'll be looking at anti-aliasing, image formats, as well as a **super resolution**, and **ray tracing denoising**.

References

[1] James T Kajiya. "The rendering equation". In: Proceedings of the 13th annual conference on Computer graphics and interactive techniques. 1986, pp. 143–150.