IMGD 3000: *Basic Computer Graphics*

William DiSanto

Computer Science Dept.
Worcester Polytechnic Institute (WPI)

Overview

- Extremely over-simplified view of graphics (60 min)
- Purpose of Computer Graphics in a Game Engine
- Representations of Data
  - Geometry
  - Light
  - Maps
- Rendering
- Courses

Purpose

- A first look at:
  - Some kinds of graphical information games require
  - Discussion on why some simplifications are made
  - Free engines
  - Where to find out more
  - Some examples in games if we have time

Rendering Equation

$$L_o(\mathbf{x},\omega,\lambda,t) = L_e(\mathbf{x},\omega,\lambda,t) + \int_\Omega f_r(\mathbf{x},\omega',\omega,\lambda,t) L_i(\mathbf{x},\omega',\lambda,t)(-\omega'\cdot\mathbf{n})d\omega'$$

- Games/Real Time rendering:
  - Find ways to simplify the rendering equation
  - Target ~30+ frames per second
- We will glance at a small portion of the devices used in making game images realistic or at least appealing.

Rendering Equation

Find the amount of light in some direction w to some point x on a surface.

Integrate over the hemisphere surrounding the point x.

Incoming light for a direction w'

$$L_o(\mathbf{x},\omega,\lambda,t) = L_e(\mathbf{x},\omega,\lambda,t) + \int_\Omega f_r(\mathbf{x},\omega',\omega,\lambda,t) L_i(\mathbf{x},\omega',\lambda,t)(-\omega'\cdot\mathbf{n})d\omega'$$

The point x may itself emit light.

Lambert's Law

Consider, the material at point x may alter the reflected light in some way for different pairs of input and output directions.

Integrate with small solid angles.

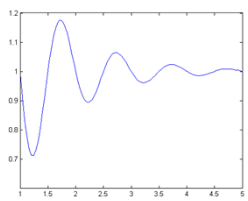
Some Representations of Data

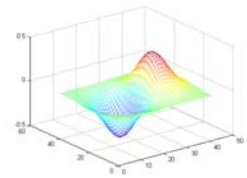
- Following slides present objects that have the following attributes:
  - All are used in modern games/engines
  - Relatively quick and easy to compute
  - Can be fitted to real world data within some measure of accuracy
  - Cannot necessarily represent real world data exactly
  - For now consider the data to represent some solid object

$$L_o(\mathbf{x},\omega,\lambda,t) = L_e(\mathbf{x},\omega,\lambda,t) + \int_\Omega f_r(\mathbf{x},\omega',\omega,\lambda,t) L_i(\mathbf{x},\omega',\lambda,t)(-\omega'\cdot\mathbf{n})d\omega'$$

Exact

- Use equations, parametric functions, etc.
- Can be evaluated at arbitrary degree of accuracy
  - Remember that your graphics card has limited accuracy





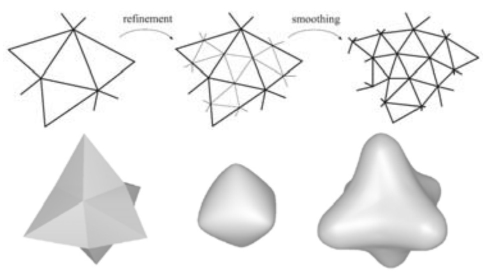
Mesh

- Mesh: connected set of vertices
- Maintained as lists of connections and vertex locations

- Compact and malleable
- Used to establish boundaries in the game world

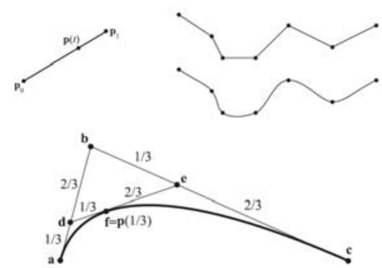
Subdivision Surface

- Insert new vertices and smooth



Spline

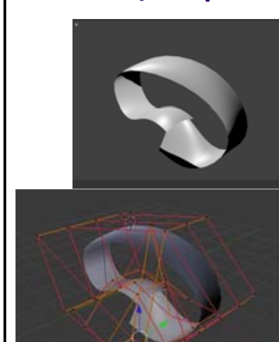
- Splines: interpolate around a set of control points
- Defined parametrically



NURBS / B-Spline

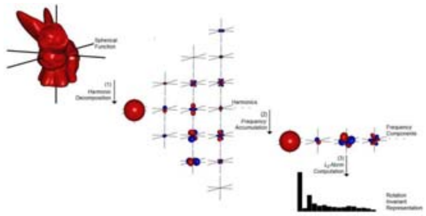
- NURBS/B-Splines
  - Local Editing
  - Transform Invariant
  - Control over continuity
  - Well defined derivatives, normals, position
  - Some editing techniques are expensive
  - Can represent conics
  - The NURBS Book

NURBS / B-Spline



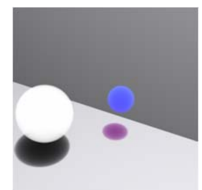
Frequency

- Decompose data into collection of equivalent scaled/modified basis functions
  - Performance gain for certain operations
  - Expensive to represent high detail objects



Probabilistic Basis Function

- Decompose data into collection of equivalent probability density functions
  - Inexpensive
  - Can model solid objects, dense objects, nebulous media



Density Grid

- Density data contained in a 3D grid
- Many ways to render

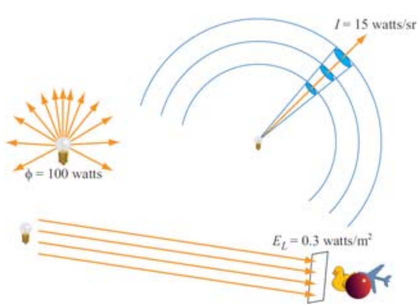
Rendered with PBRT

Light Models

- Different ways of representing light sources of different shape and distance from rendering area.
- Shape can be defined with previously mentioned representations
- Simplified model of light used to allow faster rendering

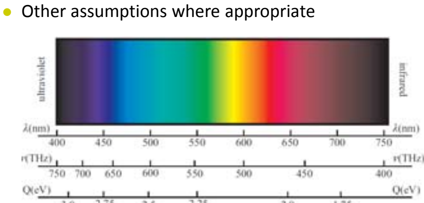
$$L_o(\mathbf{x},\omega,\lambda,t) = L_e(\mathbf{x},\omega,\lambda,t) + \int_{\Omega} f_r(\mathbf{x},\omega',\omega,\lambda,t) L_i(\mathbf{x},\omega',\lambda,t)(-\omega'\cdot\mathbf{n})d\omega'$$

Measure of Energy



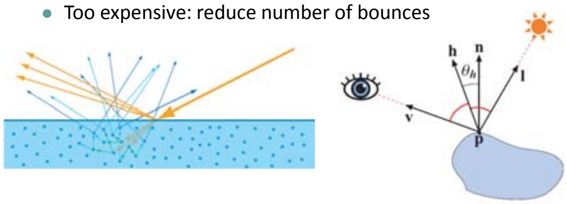
Light

- Typical Simplifying Assumptions:
  - Unpolorized
  - Sample few wavelengths
  - Speed is not considered (considered with refraction)
  - Other assumptions where appropriate



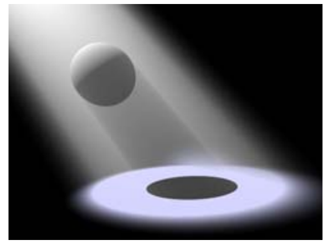
Propagation of Light

- Rays of light will reflect and refract depending on the nature of the object they hit
- More rays produced, each go on to reflect and refract off other surfaces
- Too expensive: reduce number of bounces



Spot + Directional

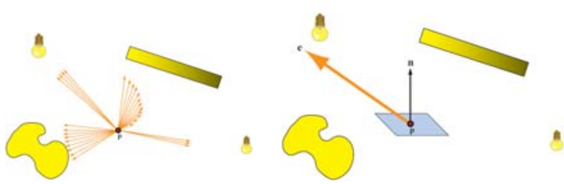
- Light originates at a point or a infinite plane and moves in a direction (all rays in directional case are parallel)



Spot Lighting PBRT: Not Real Time

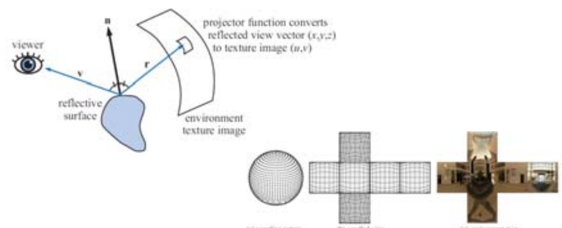
Area Lighting

- Light originates from many locations on a surface
- Many surfaces too complicated to integrate directly
- Sample or average to increase frame rate



Environment Lighting

- Light originates from many locations on a surface
- Many surfaces too complicated to integrate directly
- Sample or average to increase frame rate



Mappings

- OpenGL DirectX support 2D + 3D textures
- Textures can be used to map information to other objects in ways that can improve rendering efficiency.
  - Require the mapped-to object to store map coordinates
- They can be viewed as objects themselves:
  - Vertex/index data can be written to texture
  - Can be used as flat sprites
  - 3D textures as voxel grids (density/opacity)
  - Can model any part of the rendering function

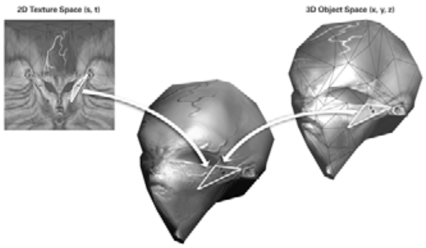
$$L_o(\mathbf{x},\omega,\lambda,t) = L_e(\mathbf{x},\omega,\lambda,t) + \int_\Omega f_r(\mathbf{x},\omega',\omega,\lambda,t)L_i(\mathbf{x},\omega',\lambda,t)(-\omega'\cdot\mathbf{n})d\omega'$$

Albedo Texturing

- Set or modify the color of an object
- Map pixels in texture to surface of triangles

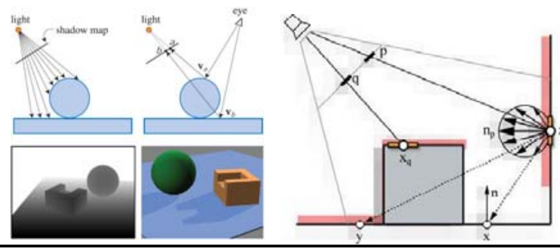
Bump/Normal Mapping

- Set or modify the normal of an object



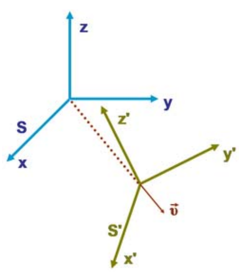
Shadow/RSM

- Render scene from perspective of light source
- Use pixels to indicate regions that receive light or generate indirect illumination



Transformations

- Matrix transformation of:
  - Control Points
  - Vertices, Normals, directions
  - Centers
  - Other matrix transforms
- Rotate, Scale, Translate, Project, …
- Performed on the GPU

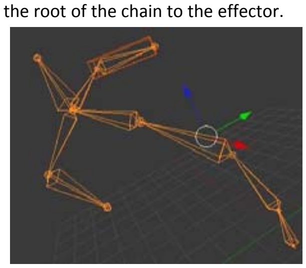


Animation

- An area of study in its own right.
  - Important to CG since animation transformations may take place on the graphics hardware
  - We will ignore that animations:
  - Can collide
  - Have mass and acceleration
  - Limited degrees of translational and rotational freedom
  - Etc.

Forward Kinematic

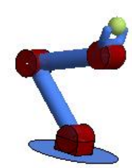
- Compute orientations of the armature
  - from the root of the chain to the effector.



http://demonstrations.wolfram.com/ForwardKinematics/

Inverse Kinematic

- Compute orientations of the armature
  - from the effector to the root
  - Many solutions

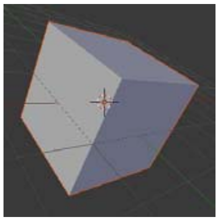


http://demonstrations.wolfram.com/InverseKinematics/

Projection: Perspective/Ortho

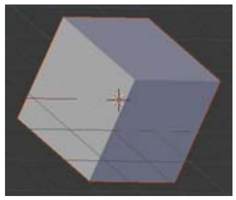
- Sample the world with rays from the camera

Rays from camera originate at a point



Perspective

Rays from camera originate at a plane



Ortho

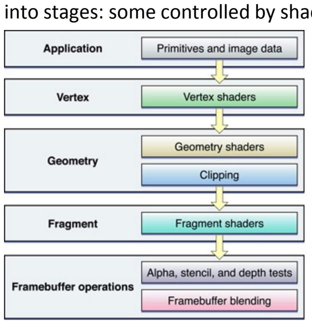
Rendering

- Sample the world over a given interval
  - Select w rays along which light energy is integrated
  - Combine observations together with other effects
    - Camera lenses distortion
    - Focus/blur
    - Pretty much anything from image processing

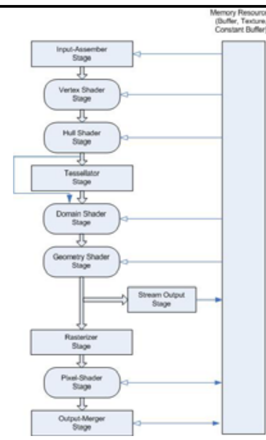
$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t)(-\omega' \cdot \mathbf{n}) d\omega'$$

Graphics Pipeline

- Broken into stages: some controlled by shader programs



DX11



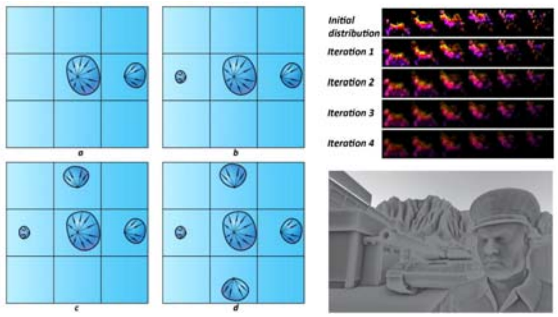
Free Engines

- Ogre
- Unity (to some degree)
- Blender
- Lux

Miscellaneous

- Real time graphics is becoming increasingly influenced by physical models
  - Most convincing renders are computed with information taken from experiment
  - Graphics hardware has progressed to the point were some ray-trace scenes are realizable with at least interactive frame rates

CryENGINE: LPV, SSAO



What was Missed?

- How to do any of these things
- How to do them in an efficient manner
- How to program with OpenGL or DirectX
- How to program shaders
- Photorealistic techniques
- A whole lot more…

Graphics Oriented Classes

- **CS 4731 Computer Graphics**
- **CS 545 Digital Image Processing**
- **CS 549 Computer Vision**
- **CS 563 Advanced Computer Graphics**
  - Courses will generally focus on aspects graphics itself rather than graphics as it applies to games in particular.
  - Signals/Systems + Applied Math Classes may help

References

- *Wan, L., Wong, T.-T., and Leung, C.-S. (2007). Isocube: Exploiting the Cubemap Hardware. IEEE Transactions on Visualization and Computer Graphics, 13(4):720–731.*
- *Michael Kazhdan , Thomas Funkhouser , Szymon Rusinkiewicz, Rotation invariant spherical harmonic representation of 3D shape descriptors, Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing, June 23-25, 2003, Aachen, Germany*

References

- *Carsten Dachsbacher , Jan Kautz, Real-time global illumination for dynamic scenes, ACM SIGGRAPH 2009 Courses, p.1-217, August 03-07, 2009, New Orleans, Louisiana*
- http://en.wikipedia.org/wiki/Inertial_frame_of_reference (figure of reference frames)
- https://developer.apple.com/library/mac/#documentation/graphicsimaging/conceptual/OpenGL-MacProgGuide/opengl_shaders/opengl_shaders.html (shader pipeline image)

References

- *Kaplanyan, A. (2009). Light propagation volumes in cryengine 3.*
- *The CG Tutorial*
  by Randima Fernando and Mark J. Kilgard
- *Physically Based Ray Tracing*
  by Matt Pharr and Greg Humphreys
- *Real Time Rendering*
  by Tomas Akenine-Möller, Eric Haines and Naty Hoffman
- *http://msdn.microsoft.com/en-us/library/windows/desktop/ff476882%28v=vs.85%29.aspx*