CMSC740
Advanced Computer Graphics

Fall 2025

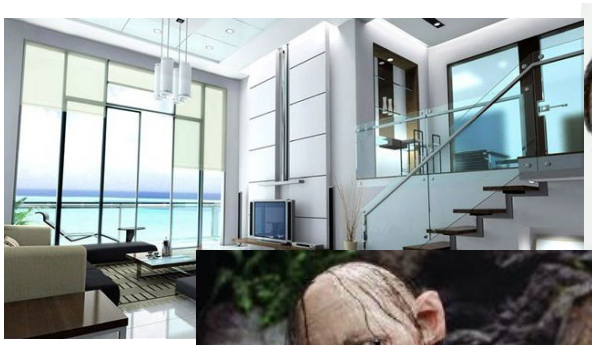
Matthias Zwicker

Today

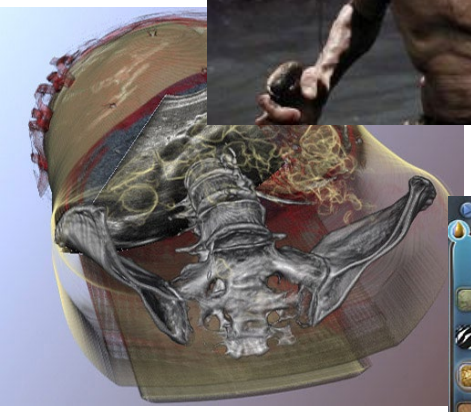
- Course overview
- Organization
- Introduction to rendering

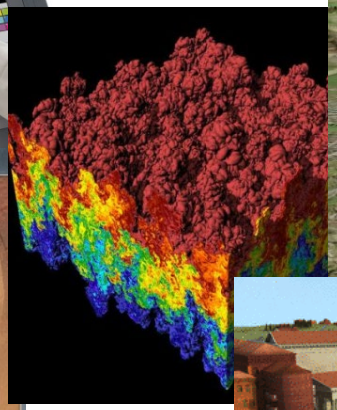
Computer graphics

"Algorithms to create and manipulate images"





Challenges

- Modeling, content creation requires intense manual work
  - 3D modeling & animation, image & video image editing & processing, etc.
- Realistic rendering computationally expensive
  - Real-time rendering in AR/VR limited to lower quality
- Currently
  - Feature length movies cost hundreds of millions
  - Limited applications of AR/VR

Research vision

- AI-based digital media creation
  - User provides simple, high-level input
  - AI system fills in details
- Benefits
  - Faster, cheaper media creation
  - Democratize visual story-telling: broaden access to low-budget producers, like small studios and individuals
  - Support and enable new media formats (AR, VR, …)

Core areas

- Rendering: computing images given 3D scene descriptions (geometry, light sources, material properties)
- Modeling: creating 3D scene descriptions (manual, algorithmic, reconstruction from measurements/images)
- Animation: making things move (physics simulation)

This course

- First half: rendering (photo-realistic)
- Second half: modeling
- Focus on recent neural-network/AI based techniques

Other graphics courses at UMD

- CMSC425, Game Programming
- CMSC427, Computer Graphics (real-time rendering)
- CMSC498F/838C: Advances in XR (AR/VR)
- CMSC828X, Physically-based Modeling, Simulation, and Animation
- CMSC848B, Computational Imaging

Today

- Course overview
- Organization
- Introduction to rendering

Organization

- Class material
  - Course page on UMD Canvas, log in with directory ID
  - Piazza for course questions
  - Virtual Study Assistant AI chatbot, configured with course materials
- Teaching assistants
  - Haiyang Yin, Yun He, CS PhD students

Organization

- Lectures
  Tuesdays and Thursdays, 2pm-3:15pm,
  IRB 2107
- Office hours
  - TAs: Mondays, 3-5pm, location TBA
  - Instructor: By appointment (send me email)

Grading

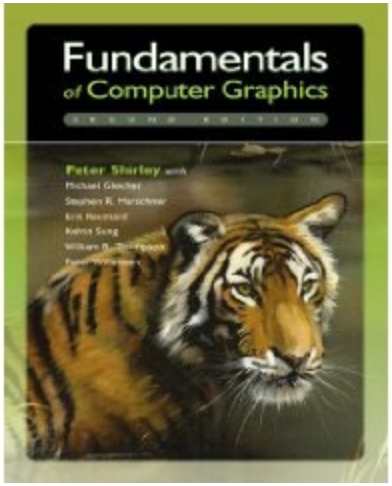
- Programming assignments
  - 40% of final grade
  - Individual projects
  - During first half of course
- Final project
  - 40% of final grade
  - Groups of two
  - During second half of course
- Final exam
  - 15% of final grade
- Class participation
  - 5% of final grade

Policies

- Standard UMD academic integrity policies
  https://www.studentconduct.umd.edu/academic-dishonesty
- Programming assignments
  - Each student submits own solution (no copying, no direct collaboration on code, no AI generated code)
  - Penalty for late submission: 50% of original score
  - Public posting (github etc.) of solutions prohibited

What you should know

- Programming experience
  in Java, Python, C++, data structures
  (CMSC420)
- Linear algebra (MATH240)
  - Vectors, matrices, systems of linear equations, coordinate transformations
- (Basic background in 3D graphics)
  - E.g., Shirley, Fundamentals of Computer Graphics



What you should know

- Mathematics
  - Vectors in 3D (dot product, vector product)
  - Matrix algebra (addition, multiplication, inverse, determinant, transpose, orthogonal matrices)
  - Coordinate systems (homogeneous coordinates, change of coordinates)
  - Transformation matrices (scaling, rotation, translation, shearing)
  - Plane equations
  - Systems of linear equations (matrix representation, elementary solution procedures)
- Graphics
  - Color
  - 3D scene representation (triangle meshes, scene graphs)

Learning objectives

- Understand and apply mathematical concepts and advanced algorithms in computer graphics
  - Rendering (rendering equation, Monte Carlo integration, path tracing, 3D geometry processing, numerical techniques, etc.)
  - Deep learning/AI in computer graphics
- Sharpen your programming skills (python, neural network techniques)

Python programming experience

A. Never used
B. Beginner (worked with it less than 16 hours/2 days)
C. Intermediate (worked with it 16-80 hours/2-10 days)
D. Advanced

I have trained a neural network

A. Yes

B. No

I can explain matrix multiplication

A. Strongly Agree
B. Agree
C. Somewhat Agree
D. Neutral
E. Somewhat Disagree
F. Disagree
G. Strongly Disagree

I can explain the concept of the basis of a vector space

A. Strongly Agree
B. Agree
C. Somewhat Agree
D. Neutral
E. Somewhat Disagree
F. Disagree
G. Strongly Disagree

Today

- Course overview
- Organization
- Introduction to rendering

First half of course

- 7-week crash course “theory and practice of photo-realistic rendering for computer graphics”

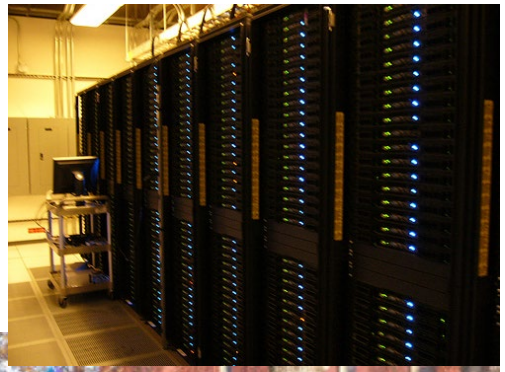
Rendering algorithms

- Problem statement
  - Given computer representation of 3D scene, generate image of the scene that would be captured by a virtual camera at a given location
- Applications
  - Movies, games, design, architecture, virtual/augmented reality, visualization, tele-collaboration, etc.

Challenges

- „Photo-realism“
  - Realistic simulation of light transport

- Efficiency
  - Thousands of CPU years for movie production

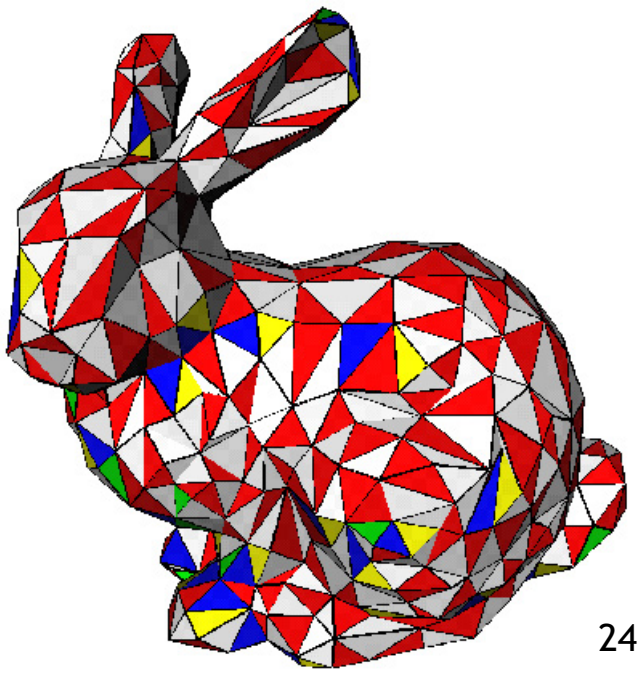


Challenges

- Real-time rendering
  - Games, VR, AR (90 frames per second (fps), 7-15ms motion-to-photon latency)
  - Highly limited computation budget for light transport simulation

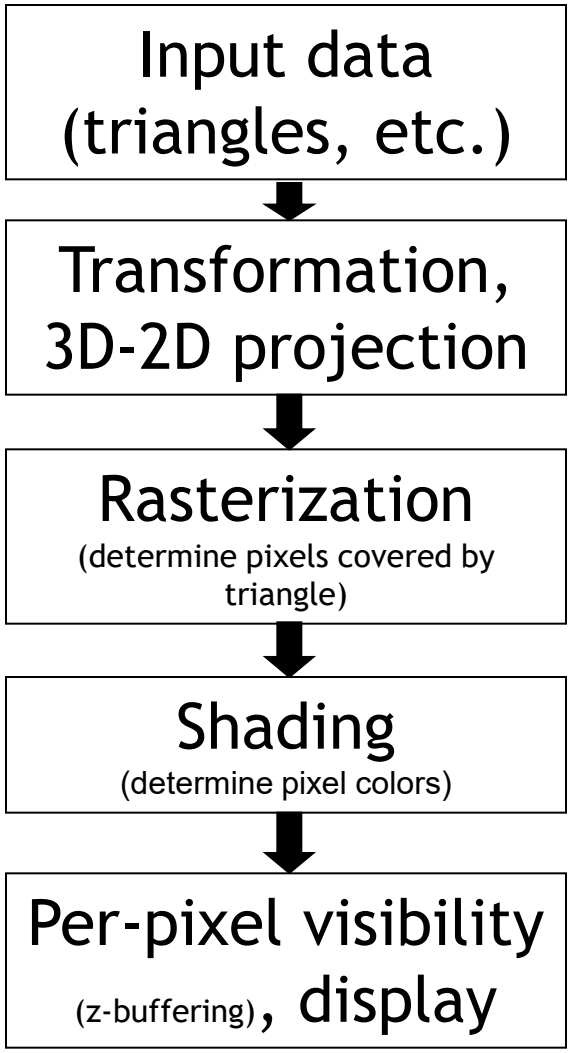
Rendering strategies

- Two fundamentally different strategies for image generation (rendering)
  1. Rendering pipeline (Z-buffering, rasterization)
  2. Ray tracing
- In both cases, most popular object representation are triangle meshes
  - Data structure: array of x,y,z vertex positions; index array indicating triplets of vertices forming triangles



Rendering pipeline

https://en.wikipedia.org/wiki/Graphics_pipeline



Object order algorithm, paint (transform, project, rasterize) **one primitive (usually triangle) after another**, visibility via z-buffer (store/compare per-pixel depth)

Z-buffering

- Implemented in graphics hardware (ATI, NVidia GPUs)
- Standardized APIs (OpenGL, Direct3D, Vulcan)
- Interactive rendering, AR/VR, games
- Limited photo-realism

Ray tracing

Output image

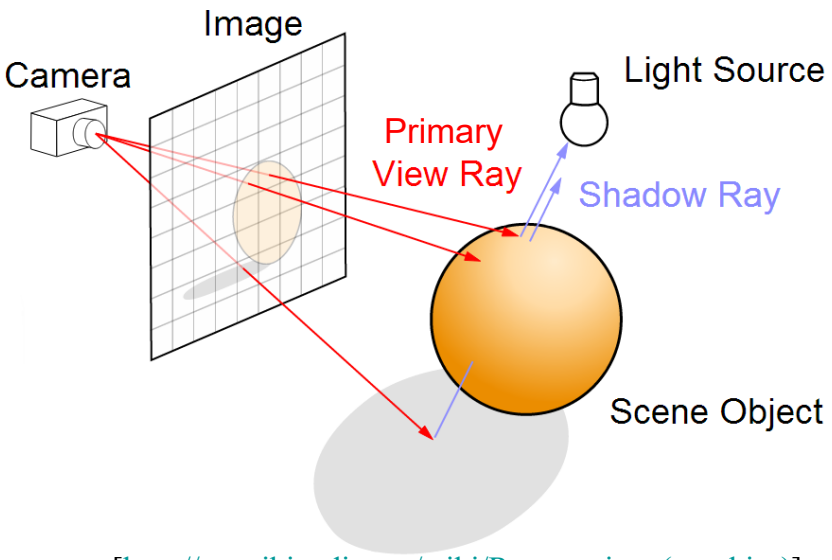
Imager (loop over pixels)

Primary view ray for each pixel

Find intersection with scene

Shading (using shadow ray)

3D scene representation



[http://en.wikipedia.org/wiki/Ray_tracing_(graphics)]

Image order algorithm, paint **one pixel after another**

Ray tracing vs. rasterization

- Rasterization
  - Desirable memory access pattern („streaming“, data locality, avoids random scene access)
  - Suitable for real time rendering (OpenGL, DirectX)
  - Full global illumination not possible with purely object order algorithm
- Ray tracing
  - Undesirable memory access pattern (random scene access)
  - Requires sophisticated data structures for fast scene access
  - Full global illumination possible
  - Most popular for photo-realistic rendering
  - Recently with hardware support

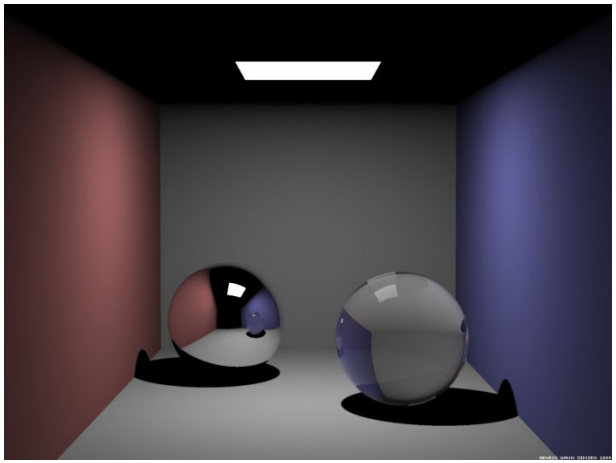
https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

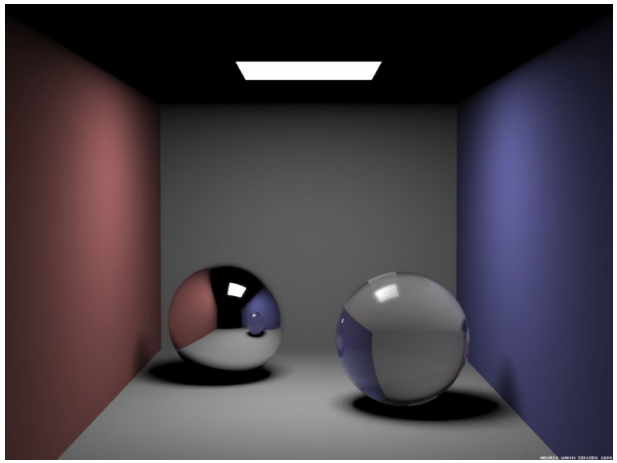
Goal of first half of class

- Realistic rendering: learn theory and practice of light transport simulation for computer graphics
  - "Rendering algorithms" based on ray tracing
  - Spatial data structures for ray tracing, physics background, rendering equation, (quasi) Monte Carlo integration, (multiple) importance sampling, Markov Chain Monte Carlo sampling, leveraging neural networks, etc.

Realistic rendering

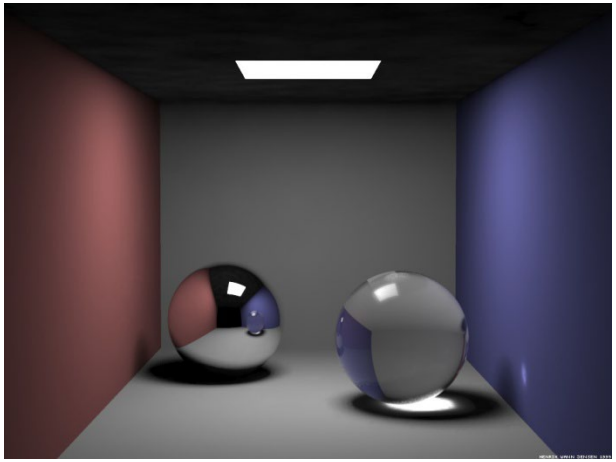
Ray tracing

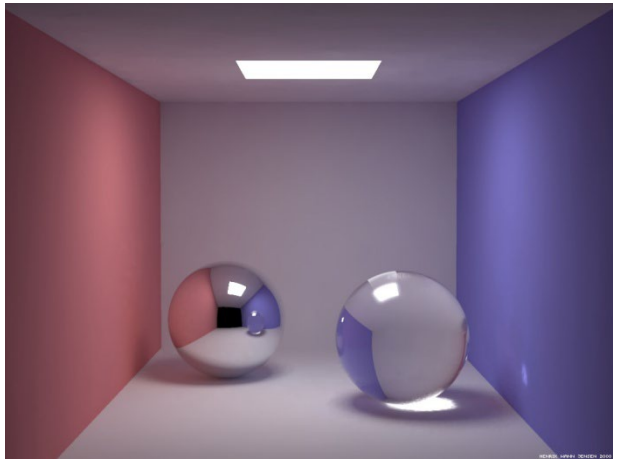




Soft shadows

Caustics

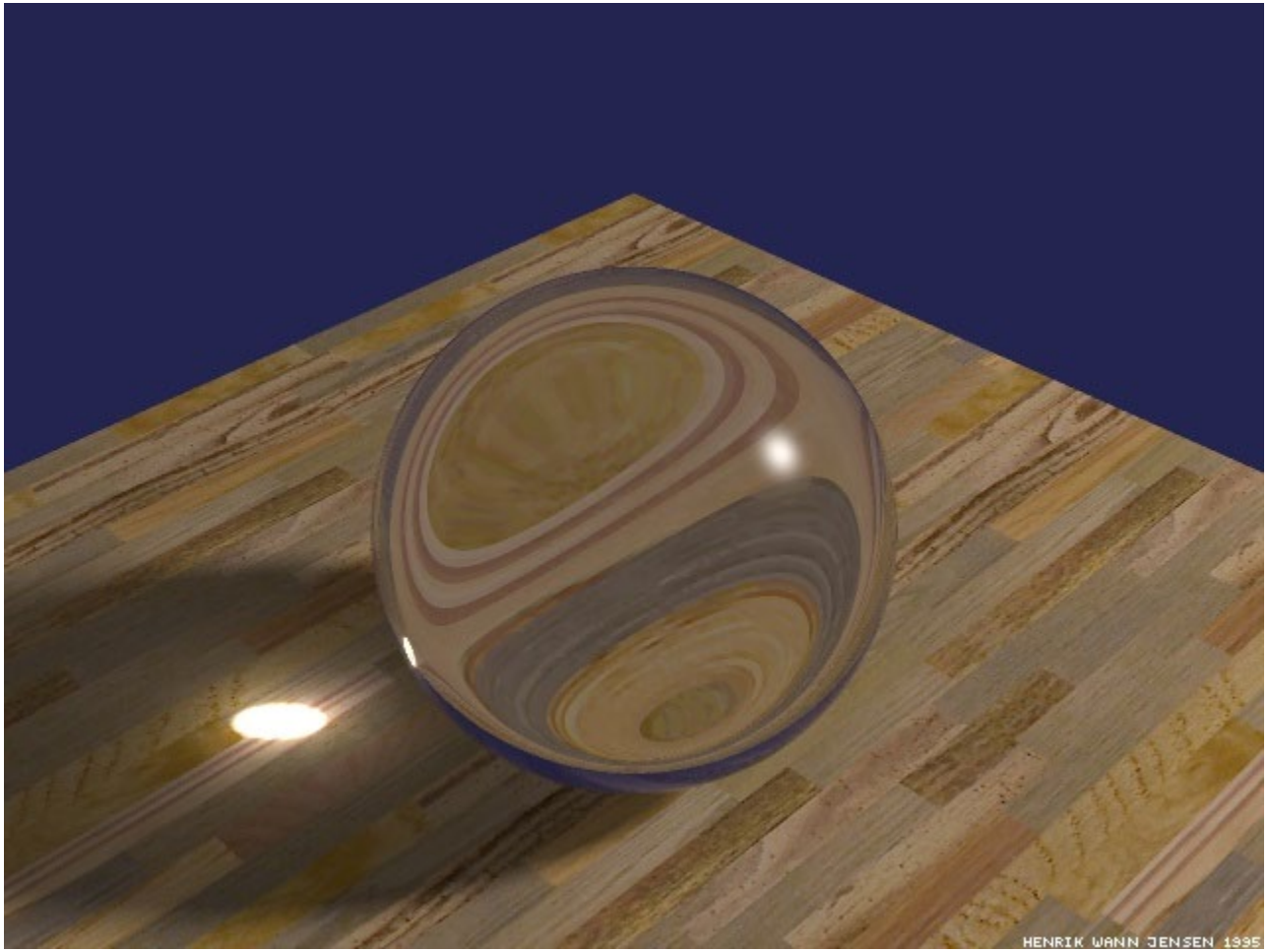




Global illum.

[Wann Jensen]

Caustics

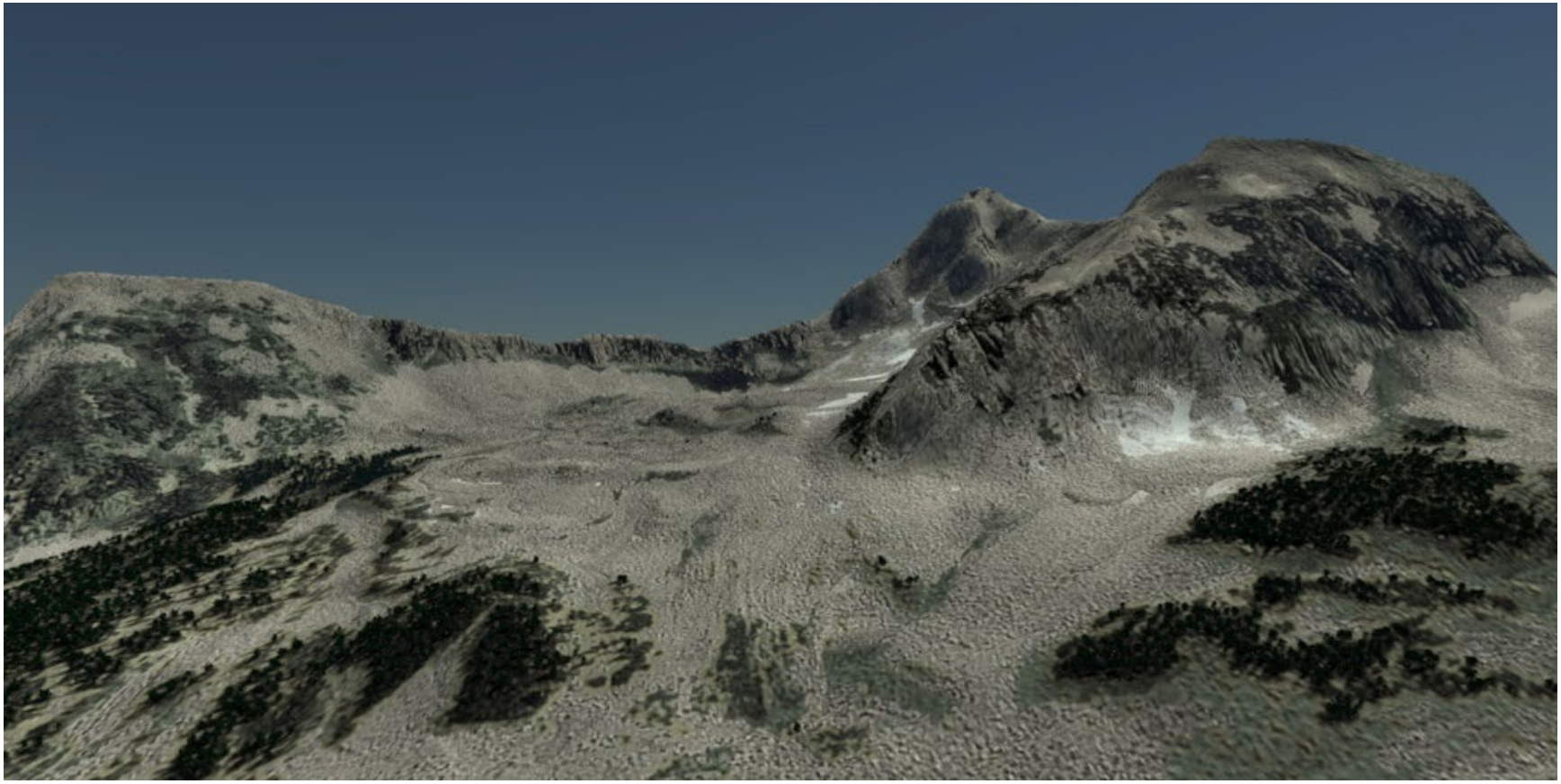


[Wann Jensen]
http://graphics.ucsd.edu/~henrik/papers/book/

Indirect illumination

[Wann Jensen]

Atmospheric effects

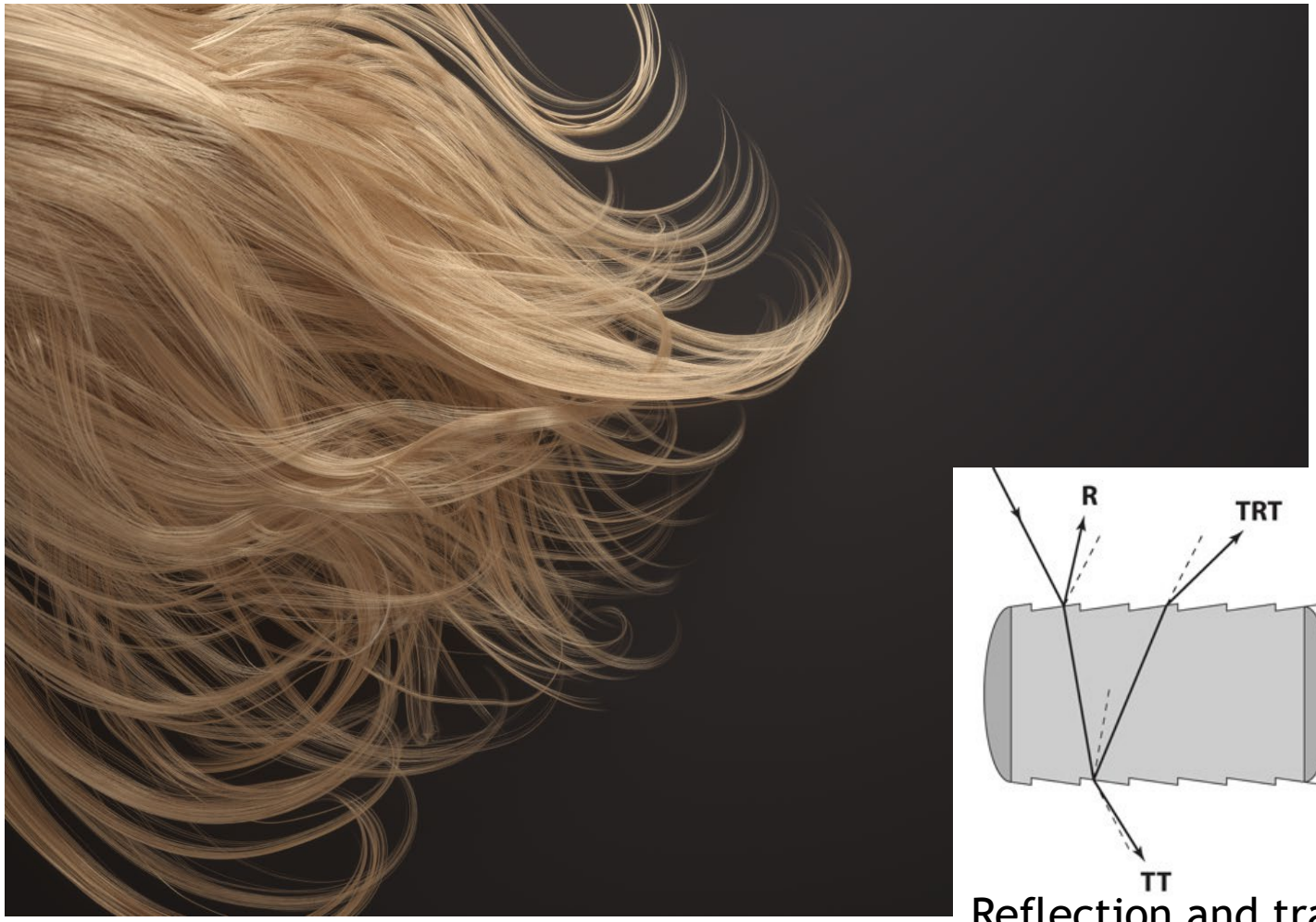


[Wann Jensen]
http://graphics.ucsd.edu/~henrik/papers/nightsky/

Participating media

[Novak et al.]
http://zurich.disneyresearch.com/~wjarosz/publications/novak12vrls.html

Complex materials



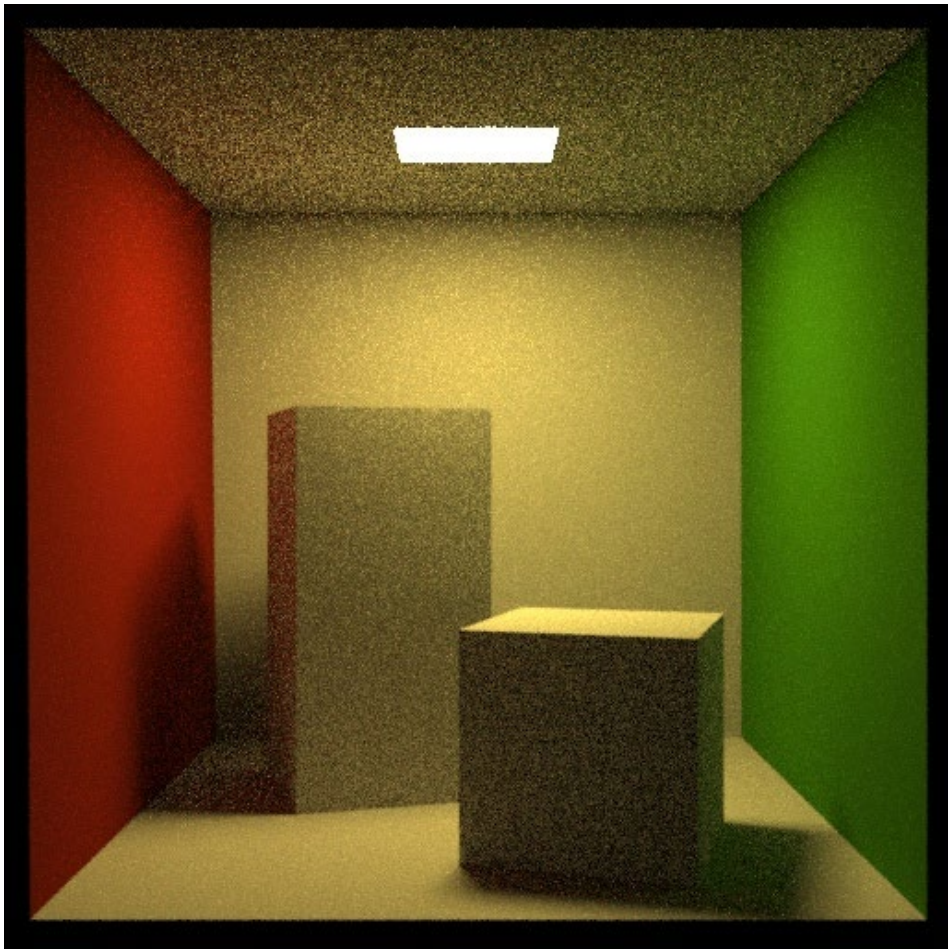
Maxwell renderer

http://www.maxwellrender.com/index.php/features

Reflection and transmission in a hair fiber

[http://www.cs.cornell.edu/~srm/publications/SG03-hair-abstract.html]

Rendering artifacts, noise



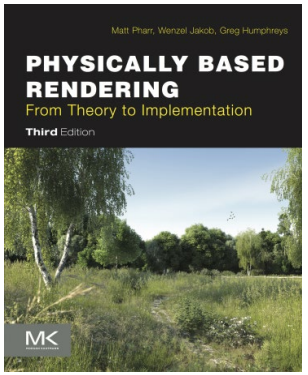
Cornell box

http://en.wikipedia.org/wiki/Cornell_Box

Big challenge, address with more computation time, and sophisticatd algorithms

Recommended book

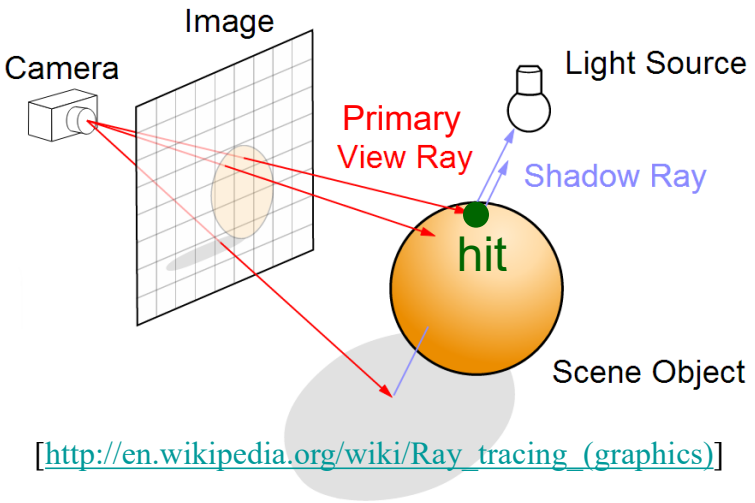
- „Physically based rendering“, by Pharr, Jakob, Humphreys
  - „Bible“ of realistic rendering
  - Available freely online
    https://www.pbr-book.org/
    http://pbrt.org/
  - Detailed, useful as reference
  - Includes many advanced concepts
  - With C++ code



Ray tracing pseudocode

```
rayTrace() {
  construct scene representation

  for each pixel
    ray = computePrimaryViewRay( pixel )
    hit = first intersection of ray with scene
    color = shade( hit ) // using shadow ray
    set pixel color
}
```



[http://en.wikipedia.org/wiki/Ray_tracing_(graphics)]

Computational complexity of ray tracing depends on

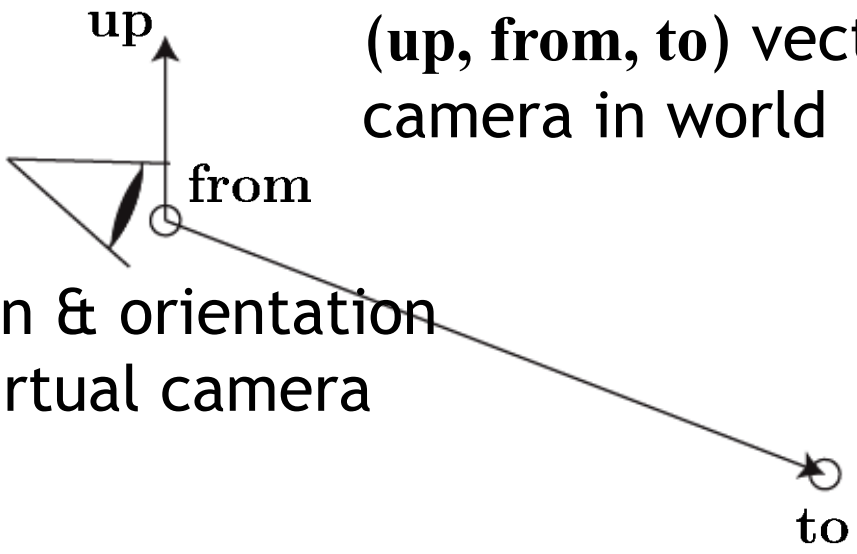
A. Number of pixels in image
B. Number of triangles in 3D triangle meshes
C. Product of number of pixels and number of triangles

Computing primary rays

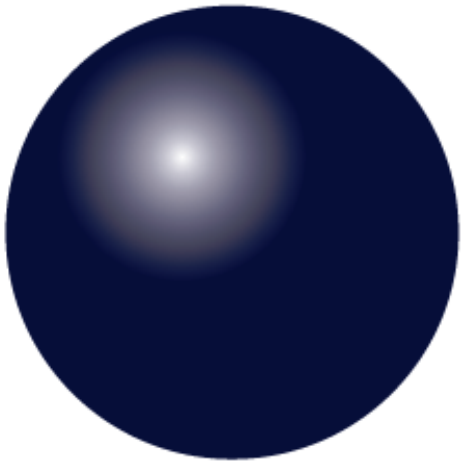
- Primary rays sometimes called “camera rays”, “view rays”
- Need to define camera coordinate system
  - Specifies position and orientation of virtual camera in 3D space
- Need to define viewing frustum
  - Pyramid shaped volume that contains 3D volume seen by camera
- Given an image pixel
  1. Determine ray in camera coordinates, intrinsic camera matrix
  2. Transform ray to world coordinates (sometimes also called “canonic” coordinates), extrinsic camera matrix

Camera coordinate system

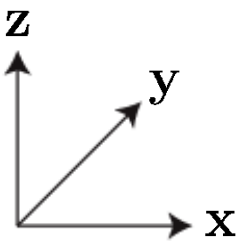
Intuitive user provided parameters (**up, from, to**) vectors to place camera in world



Position & orientation of virtual camera



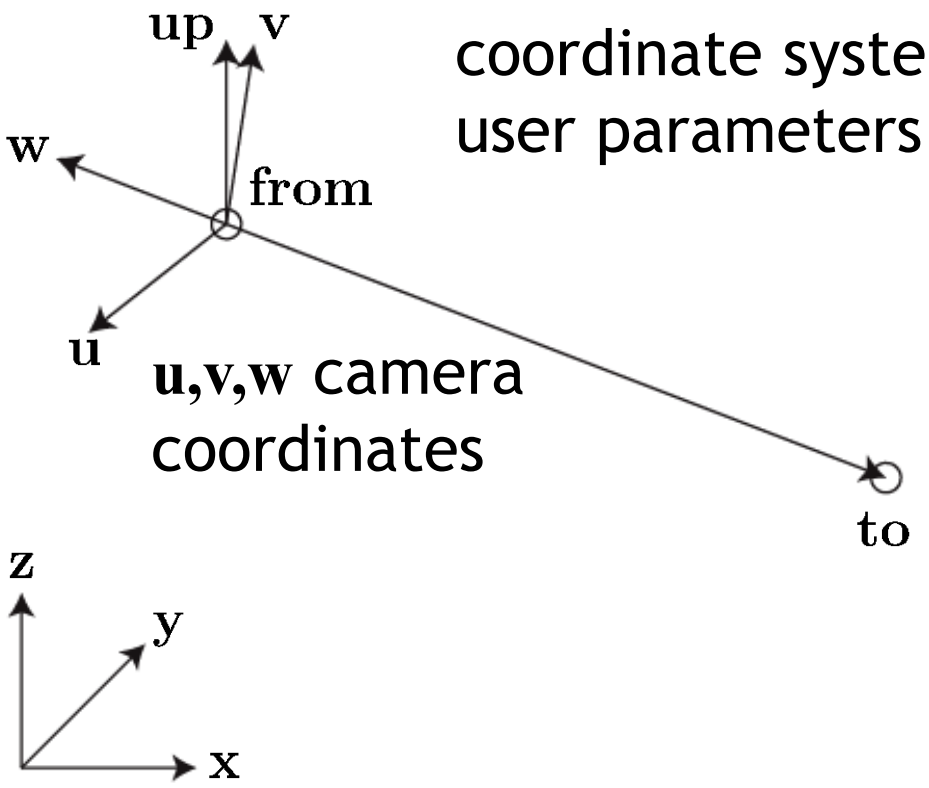
Scene object

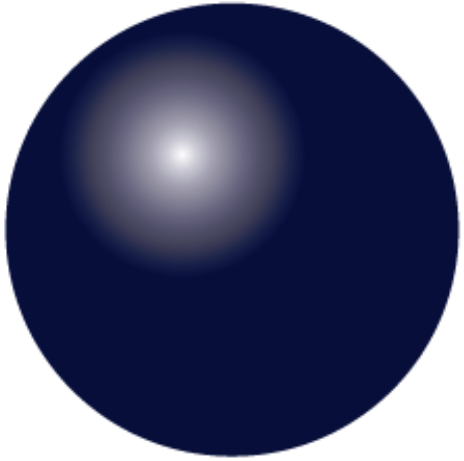


World coordinates given by **x,y,z** basis vectors

Camera coordinate system

Need to construct basis of camera coordinate system (**u,v,w**) based on user parameters (**up, from, to**)





Scene object

Right handed coordinate systems!

https://en.wikipedia.org/wiki/Right-hand_rule

Camera coordinate system

- Given **from, to, up**

$$\mathbf{w} = \frac{\mathbf{from} - \mathbf{to}}{\|\mathbf{from} - \mathbf{to}\|}$$

$$\mathbf{u} = \frac{\mathbf{up} \times \mathbf{w}}{\|\mathbf{up} \times \mathbf{w}\|}$$

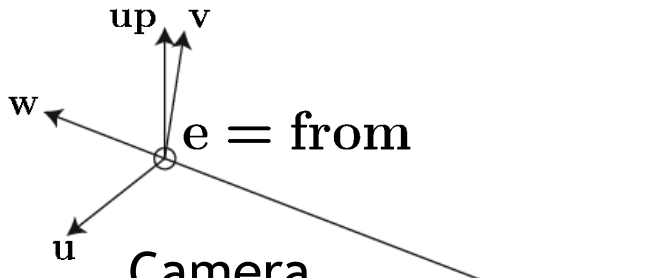
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

$$\mathbf{e} = \mathbf{from}$$

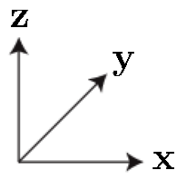
- Extrinsic 4x4 camera matrix $[\mathbf{u}\ \mathbf{v}\ \mathbf{w}\ \mathbf{e}]$

Use **homogeneous coordinates** to enable affine transformations (including rotation, translation) with matrix multiplications

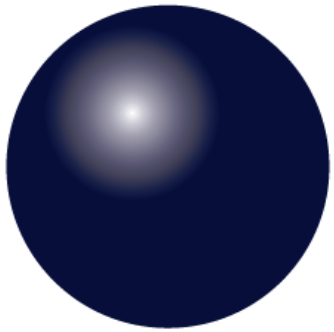
https://en.wikipedia.org/wiki/Transformation_matrix#Affine_transformations



Camera coordinates



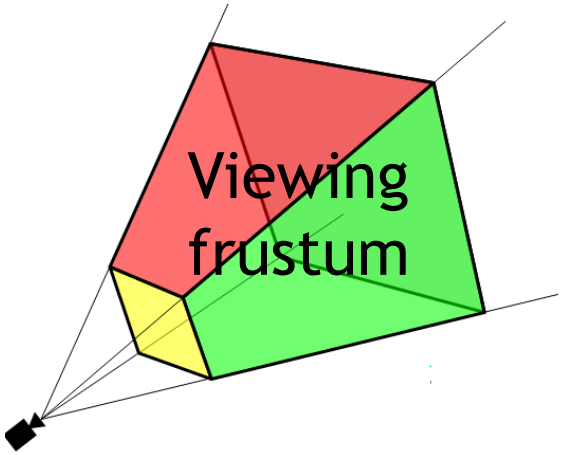
World coordinates



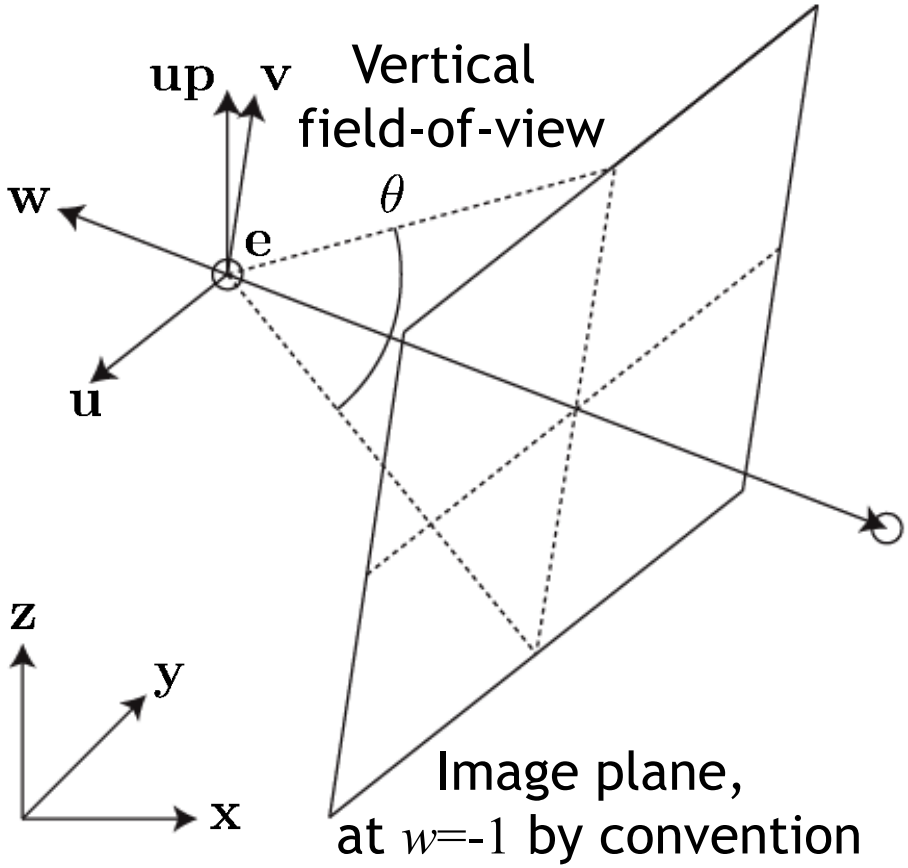
Scene object

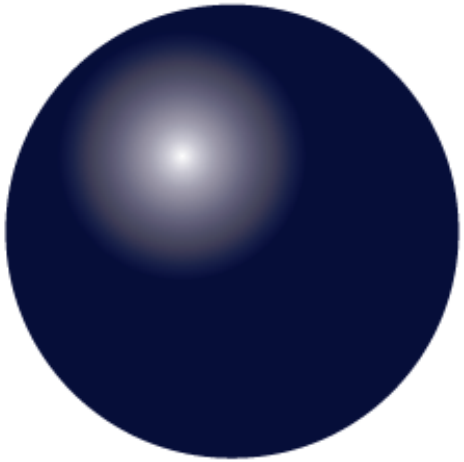
Viewing frustum

- 3D volume that is projected within image boundaries
  - "Pyramidal cone" that contains all camera rays
- Defined by
  - Vertical field-of-view θ
  - Aspect ratio $aspect = width/height$
  - By convention, image plane at w=-1

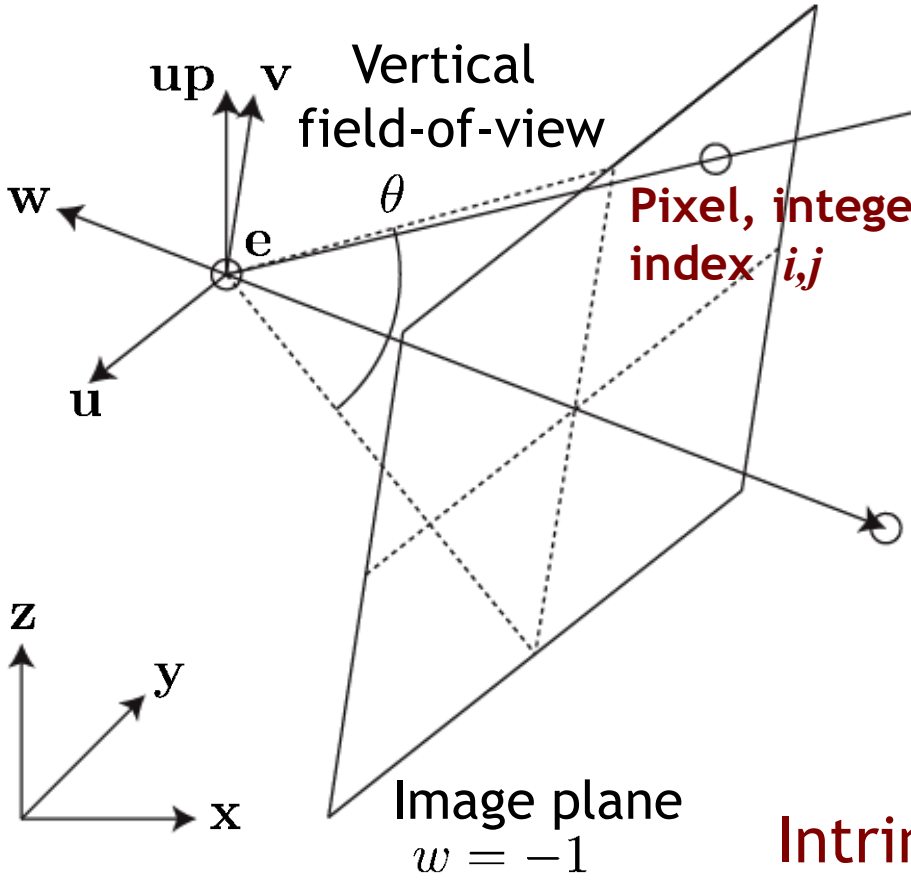


Viewing frustum



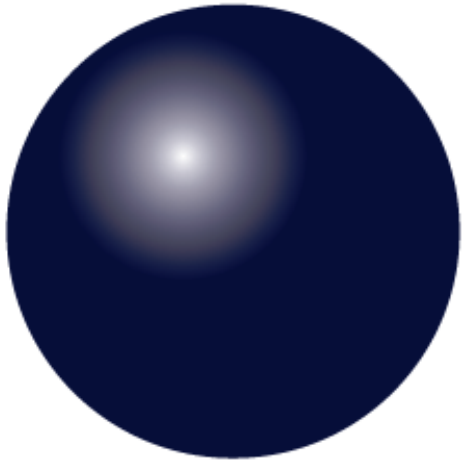


From pixel i,j to primary viewing ray



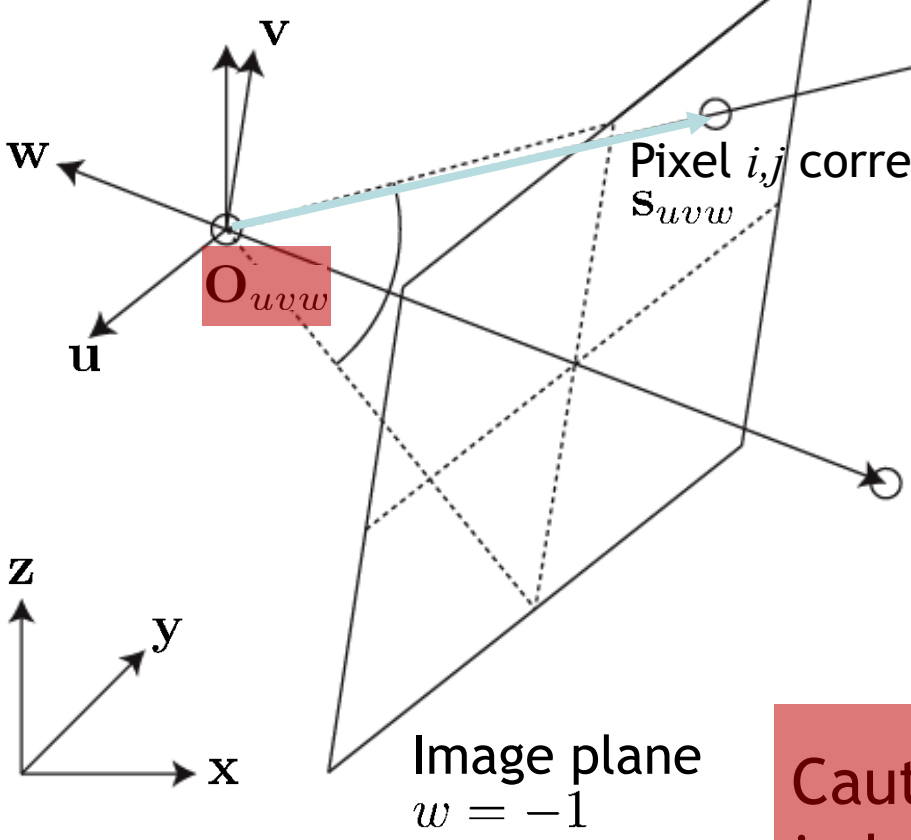
Primary viewing ray (in camera coordinates)

Pixel, integer index *i,j*



Intrinsic camera matrix K maps pixel coordinates (i,j) to ray in camera coordinates

Primary rays in camera coords.



Primary ray

$$\mathbf{p}_{uvw}(t) = \mathbf{0}_{uvw} + t(\mathbf{s}_{uvw} - \mathbf{0}_{uvw})$$

Pixel i,j corresponds to point $\mathbf{s}_{uvw}$

$$\mathbf{s}_{uvw} = \begin{bmatrix} u \\ v \\ -1 \\ 1 \end{bmatrix}$$

$\mathbf{O}_{uvw}$

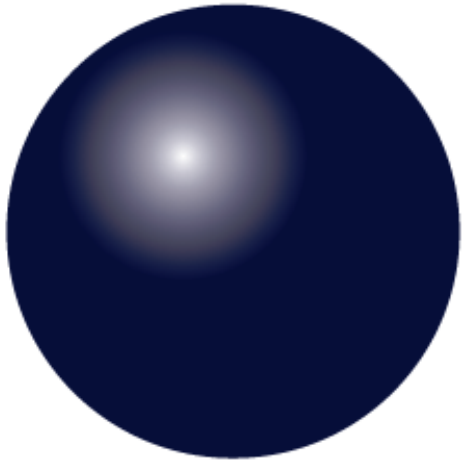
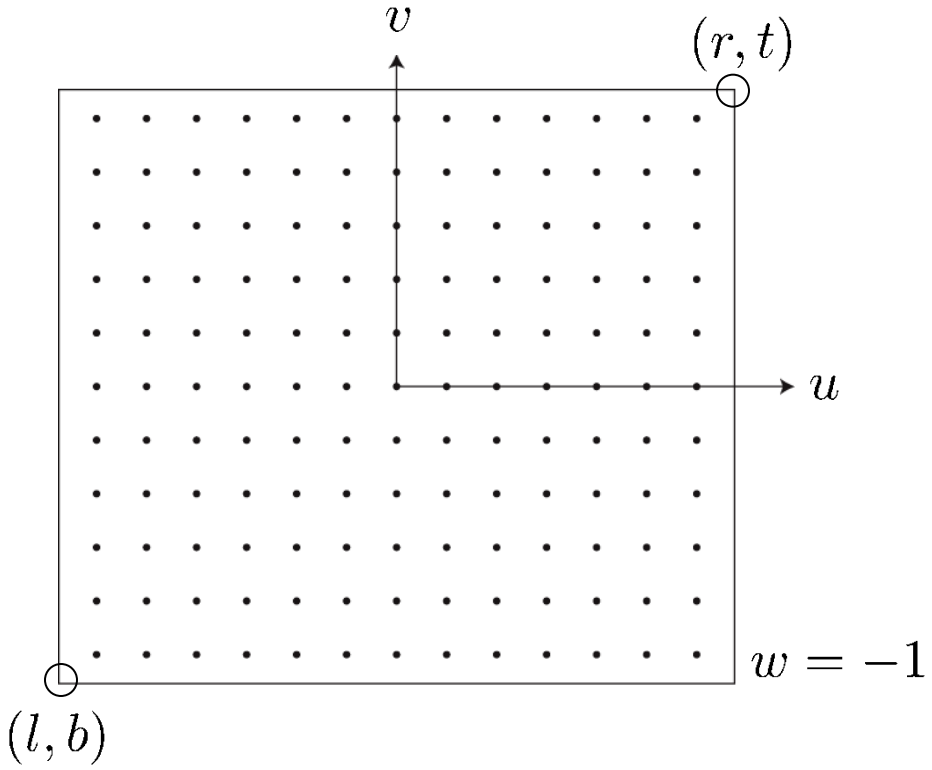


Image plane
$w = -1$

Caution: origin in homogeneous coordinates $\mathbf{0}_{uvw} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

Bounds of image plane t,b,l,r

- Vertical field-of-view θ
- Aspect ratio width/height $aspect$
- Image coordinates u, v



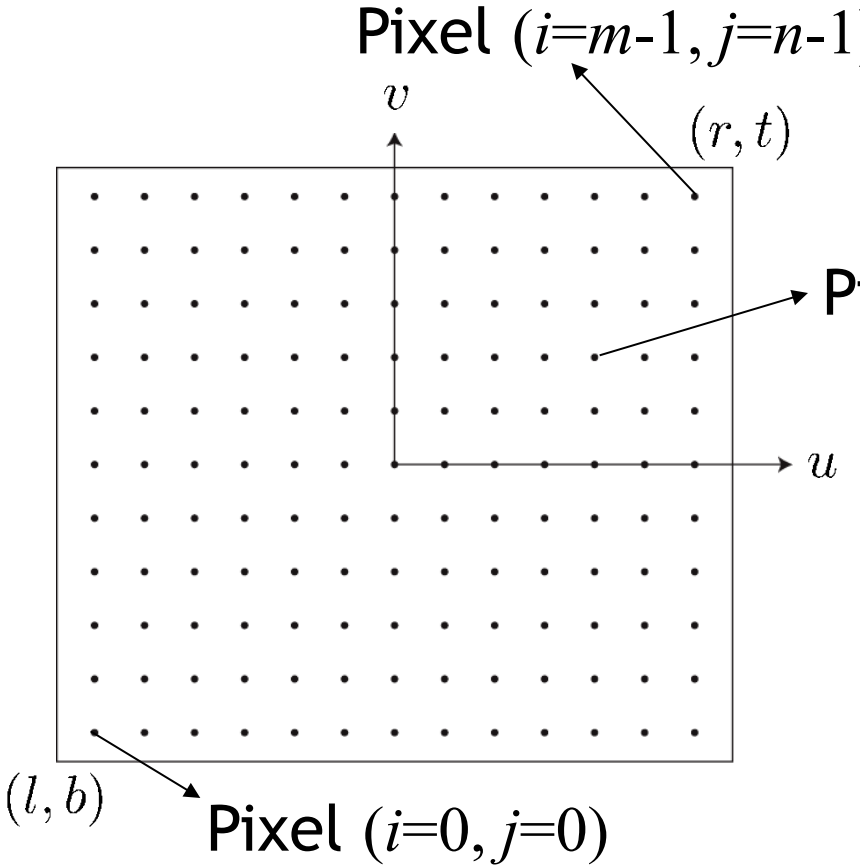
Top, bottom, left, right coords. of image corners

$$t = -b = tan(\theta/2)$$

$$r = -l = aspect \cdot t$$

Image plane position u,v given pixel index i,j

- Image resolution $m \times n$ pixels

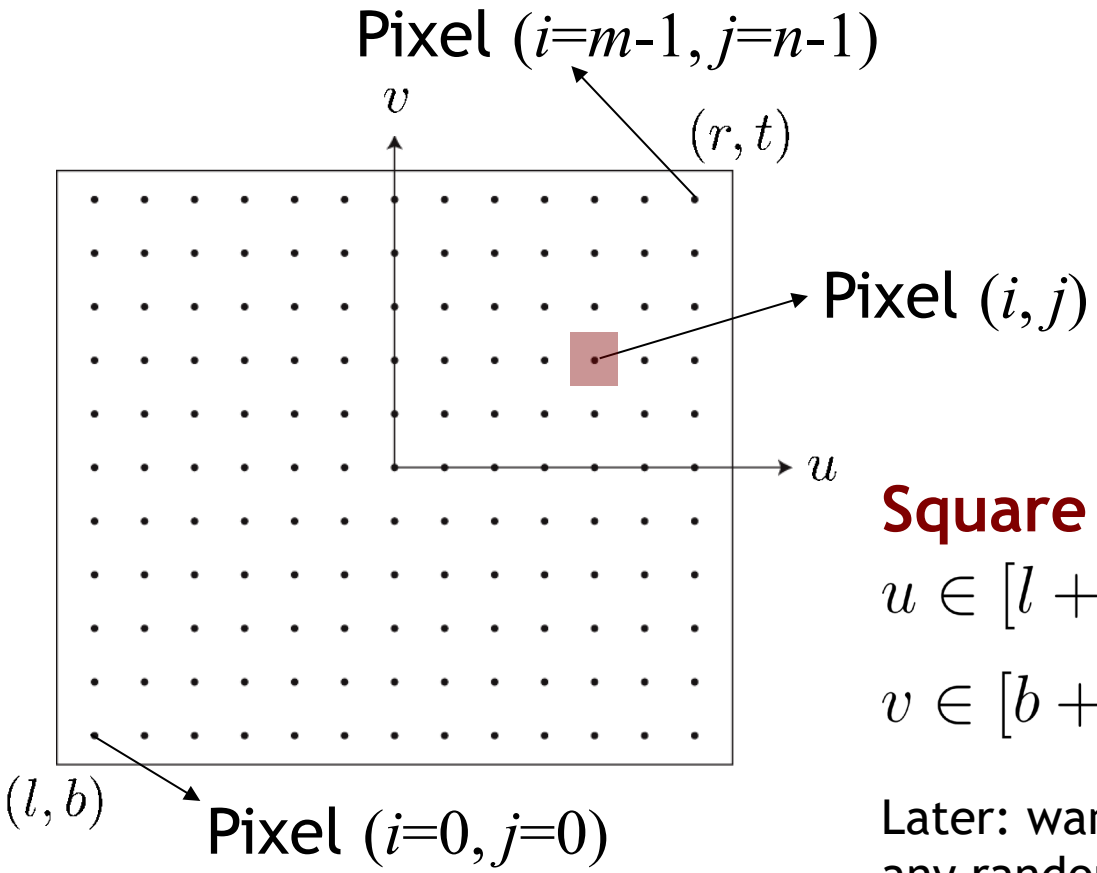


Camera coords. of **center** of pixel i, j

$$u(i,j) = l + (r-l)\frac{i+0.5}{m}$$
$$v(i,j) = b + (t-b)\frac{j+0.5}{n}$$
$$w(i,j) = -1 \quad \text{by convention}$$

Area of pixl i,j in u,v cords.

- Image resolution $m \times n$ pixels



Square area around pixel i, j

$$u \in [l + (r-l)\tfrac{i}{m}, l + (r-l)\tfrac{i+1}{m}]$$

$$v \in [b + (t-b)\tfrac{j}{n}, b + (t-b)\tfrac{j+1}{n}]$$

Later: want primary ray for pixel through any random position within pixel, not just pixel center

Primary rays in camera coords.

- Given location (u,v) on image plane, get ray $\mathbf{p}_{uvw}(t)$ in camera coords. through (u,v)

$$\mathbf{p}_{uvw}(t) = \mathbf{0}_{uvw} + t(\mathbf{s}_{uvw} - \mathbf{0}_{uvw})$$

$$\mathbf{s}_{uvw} - \mathbf{0}_{uvw} = \begin{bmatrix} u(i,j) \\ v(i,j) \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= K \begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix}$$

Intrinsic camera matrix K

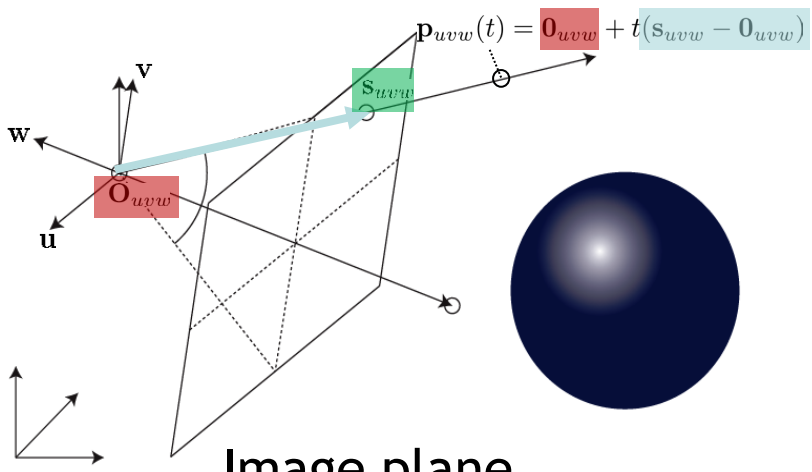
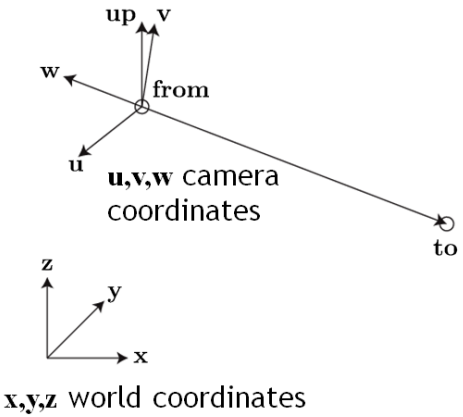


Image plane

Transformation to world coords.

- World coordinates also called “canonic” coordinates
- Basis vectors of camera coords: **u,v,w,e** column vectors
- Change of basis matrix (camera to world, extrinsic camera matrix)



https://en.wikipedia.org/wiki/Change_of_basis

$$\mathbf{M} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \end{bmatrix}$$

- Need ray $\mathbf{p}_{xyz}(t)$ in world coordinates

$$\begin{aligned} \mathbf{p}_{xyz}(t) &= \mathbf{M}\mathbf{p}_{uvw}(t) \\ &= \mathbf{M}\mathbf{0}_{uvw} + t(\mathbf{M}\mathbf{s}_{uvw} - \mathbf{M}\mathbf{0}_{uvw}) \end{aligned}$$

Ray tracing pseudocode

```
rayTrace() {
  construct scene representation

  for each pixel
    ray = computePrimaryViewRay( pixel )
    hit = first intersection with scene
    color = shade( hit ) // using shadow ray
    set pixel color
}
```

Python renderer notes

- Instead of using loops, code operates on tensors
  - Tensors represent entire sets of rays
- Advantages
  - Underlying implementation hides complexity of parallelization on multi-core CPUs, GPUs

Ray-surface intersection

- Implicit surfaces
- Parametric surfaces

Implicit surfaces

http://en.wikipedia.org/wiki/Surface#Extrinsically_defined_surfaces_and_embeddings

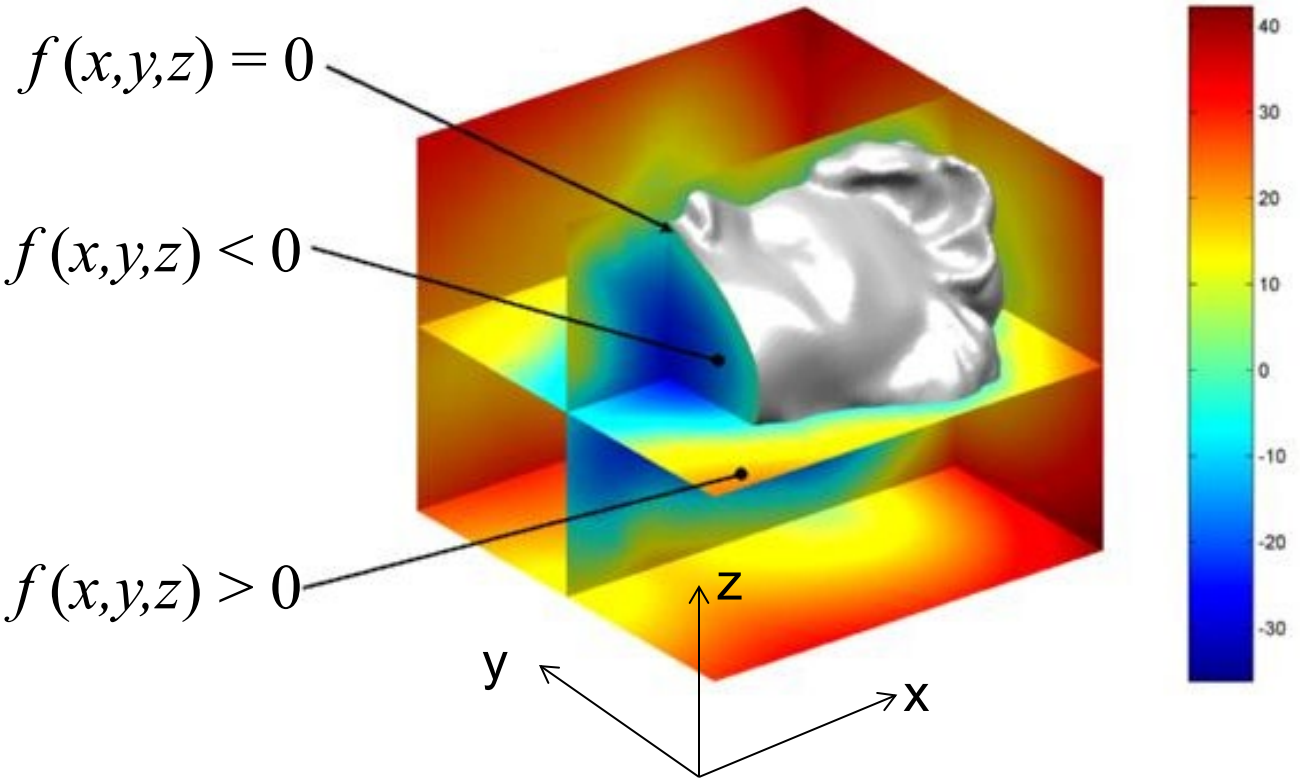
- Surface defined as set of points $\mathbf{p}$ such that

$$f(\mathbf{p}) = 0, \quad \text{where} \quad f : \mathbf{R}^3 \to \mathbf{R}, \mathbf{p} = (x, y, z)$$

- Implicit representation defines a solid
  - Including the interior volume
  - Not just the surface

Implicit surfaces

- 3D visualization, $\mathbf{p}=(x,y,z)$



Implicit surfaces

http://en.wikipedia.org/wiki/Surface#Extrinsically_defined_surfaces_and_embeddings

Ray-surface intersection

- Given a ray with origin $\mathbf{e}$, direction $\mathbf{d}$

$$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}, \quad \mathbf{p}, \mathbf{e}, \mathbf{d} \in \mathbf{R}^3, t \in \mathbf{R}$$

- Ray-surface intersection: solve for t such that

$$f(\mathbf{p}(t)) = 0$$

- In general, non-linear equation of variable t
  - Use numerical root-finding algorithm
    http://en.wikipedia.org/wiki/Root-finding_algorithm
  - For example, secant method
    http://en.wikipedia.org/wiki/Secant_method

Infinite plane

- Unit normal $\mathbf{n}$, arbitrary point on plane $\mathbf{a}$
- Plane defined as set of points $\mathbf{p}$ where

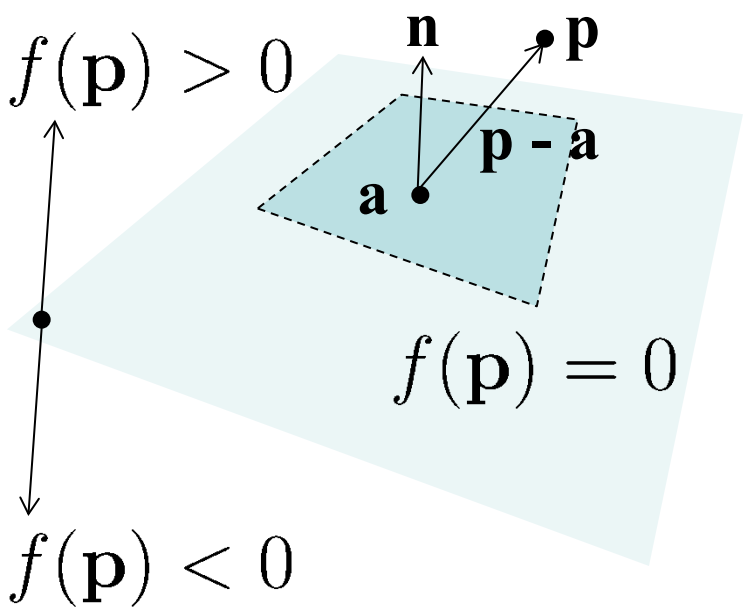
$$f(\mathbf{p}) = (\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0$$

- $f(\mathbf{p})$ is distance to plane!
- Intersection with $\mathbf{p}(t)$

$$(\mathbf{e} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} = 0$$

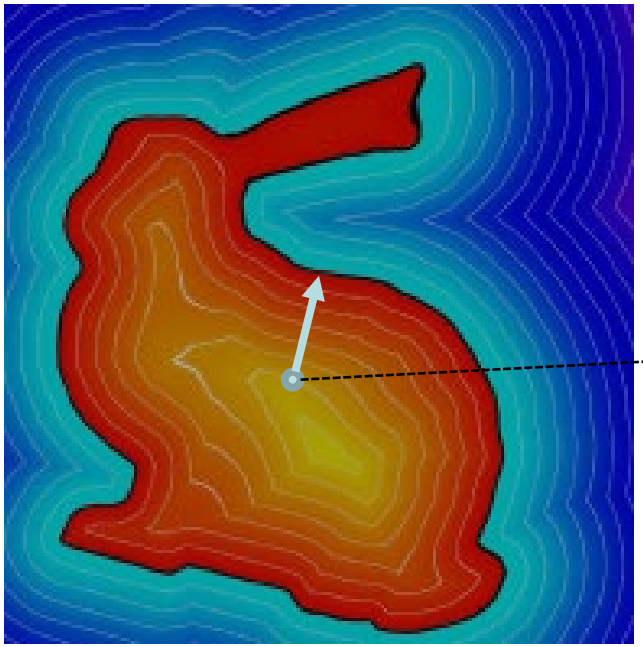
Point on ray

$$t = \frac{(\mathbf{a}-\mathbf{e})\cdot\mathbf{n}}{\mathbf{d}\cdot\mathbf{n}}$$



Signed distance function (SDF)

- Implicit representation of surfaces where function values happen to represent signed distance to closest point on surface



Parametric surfaces

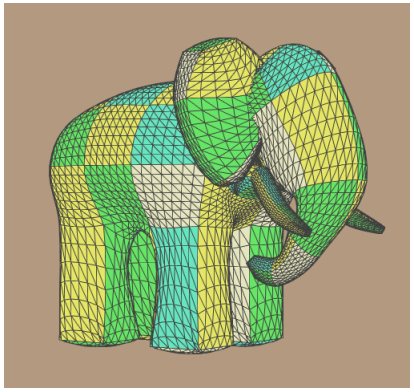
- Parametric surfaces are boundary representations (b-reps)
  http://en.wikipedia.org/wiki/Boundary_representation
  - „Solid object is given by describing its boundary between solid and non-solid“
- In general, surfaces can be described either implicitly or parametrically

Parametric surfaces

- Parameters u, v, typically in rectangular domain

$$\left.\begin{aligned} x &= f(u,v) \\ y &= g(u,v) \\ z &= h(u,v) \end{aligned}\right\} \text{ Point } \mathbf{p} \text{ on surface } \mathbf{p} = (x,y,z)$$

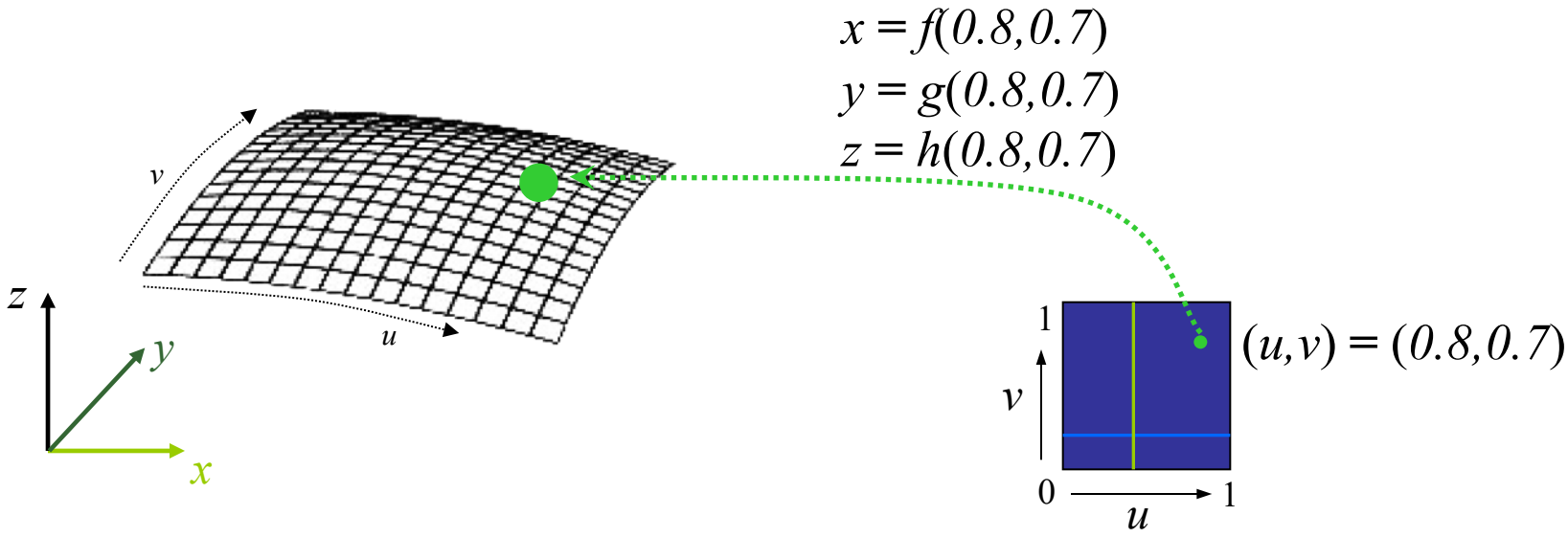
- Complicated surfaces require fitting together many parametric patches (charts)



http://en.wikipedia.org/wiki/B%C3%A9zier_surface

Parametric patches (charts)

- Visualization



2D parameter domain,
quadrilateral patch

Parametric surfaces

- Ray-surface intersection

Point on ray = point on surface

$$e_x + td_x = f(u, v)$$
$$e_y + td_y = g(u, v)$$
$$e_z + td_z = h(u, v)$$

Ray origin, direction:

$$\mathbf{e} = (e_x, e_y, e_z)$$
$$\mathbf{d} = (d_x, d_y, d_z)$$

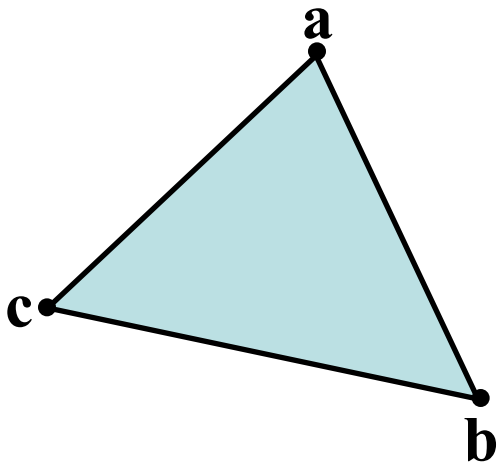
- Solve 3 equations for 3 unknowns *t, u, v*
- Easy to solve for parametric planes (see ray-triangle intersection)
- Non-linear equation for higher order surfaces (e.g., NURBS)
  - Requires iterative solution

Ray-triangle intersection

- Parametric description of a triangle with vertices **a, b, c** (parameters α,β instead of u,v)

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

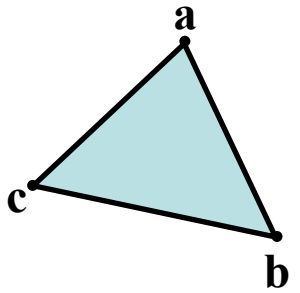
$$\text{where} \quad \beta > 0, \gamma > 0, \beta + \gamma < 1$$



- Convention: vertices are ordered counter-clockwise as seen from the outside

Ray-triangle intersection

- Outward-pointing normal



$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})$$

- Triangle in barycentric coordinates α,β,γ

$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

$$\text{where} \quad \alpha + \beta + \gamma = 1,$$

$$0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1$$

- Given two of the coordinates, can easily compute third, e.g. $\alpha = 1 - \beta - \gamma$

Ray-triangle intersection

- Intersection condition

$$\mathbf{e} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

Point on ray Point on triangle

- One equation for each coordinate x,y,z

$$e_x + td_x = a_x + \beta(b_x - a_x) + \gamma(c_x - a_x)$$
$$e_y + td_y = a_y + \beta(b_y - a_y) + \gamma(c_y - a_y)$$
$$e_z + td_z = a_z + \beta(b_z - a_z) + \gamma(c_z - a_z)$$

Ray-triangle intersection

- In matrix form

$$\begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - e_x \\ a_y - e_y \\ a_z - e_z \end{bmatrix}$$

- Solve for β, γ, t using Cramer's rule (Shirley page 157 )
- Ray intersects triangle if $0 < \beta + \gamma < 1$

Example: unit sphere

- Radius 1, centered at origin $(0,0,0)$
- Implicit
  - $f(x,y,z) = x^2 + y^2 + z^2 - 1$
- Parametric
  - Spherical coordinates (polar, azimuthal angles) θ, ϕ
  - $x = \sin(\theta)\cos(\phi)$, $y = \sin(\theta)\sin(\phi)$, $z = \cos(\theta)$

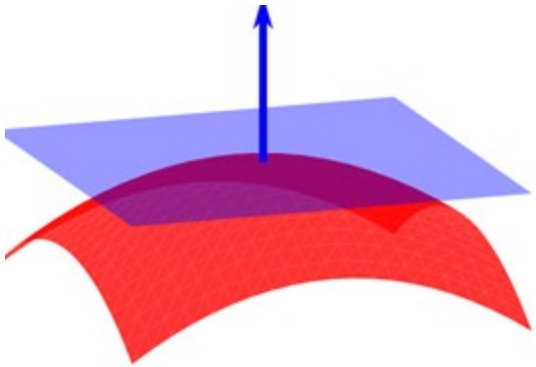
What is easier to ray trace, implicit or parametric sphere?

A. Implicit equations for sphere
B. Parametric equations for sphere

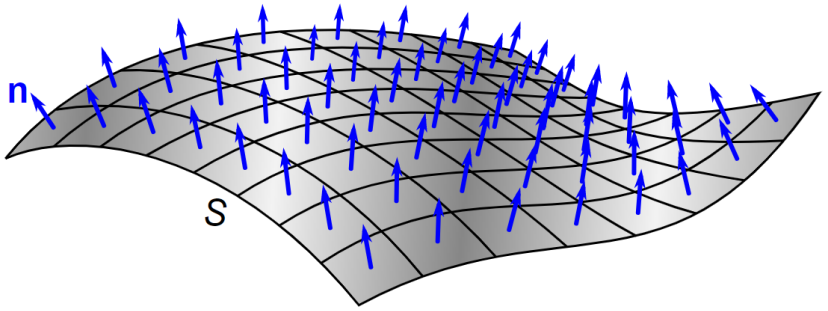
Surface normals

https://en.wikipedia.org/wiki/Normal_(geometry)

- Unit vector perpendicular to tangent plane at each point on surface (i.e., surface normal is normal of tangent plane at each point)
- Required for shading caldulations



Surface normal is perpendicular to tangent plane at each point



Normal field of curved surface

Surface normals

- How to compute for
  - Parametric surfaces?
  - Implicit surfaces?

Surface normals

- Implicit surfaces: gradient of f

$$\mathbf{n} = \nabla f(\mathbf{p}) = \left(\frac{\partial f(\mathbf{p})}{\partial x}, \frac{\partial f(\mathbf{p})}{\partial y}, \frac{\partial f(\mathbf{p})}{\partial z} \right)$$

- Parametric surface: cross product of tangent vectors (partial derivatives)

$$\mathbf{n}(u,v) = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u} \right) \times \left(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} \right)$$

Transforming Normal Vectors

- Normal $\mathbf{n}$ is perpendicular to tangent $\mathbf{t}$

$$\mathbf{n}^T \cdot \mathbf{t} = 0$$

- Under transformation $\mathbf{Mt}$, find transformation $\mathbf{X}$ of normal $\mathbf{n}$ that satisfies normal constraint

$$(\mathbf{Xn})^T \cdot \mathbf{Mt} = 0$$

$$\mathbf{n}^T \mathbf{X}^T \mathbf{Mt} = 0$$

$$\Rightarrow \mathbf{X}^T \mathbf{M} = \mathbf{I} \Rightarrow \mathbf{X} = \mathbf{M}^{-1^T}$$

- Note: $\mathbf{M}^{-1^T} = \mathbf{M}$ if and only if $\mathbf{M}$ includes only rotation and translation (no shear)

Rendering assignments

- Assignments available on UMD ELMS/Canvas
  https://myelms.umd.edu
- Extend an educational renderer based on Python and tensors
- First assignment
  - Getting started with base code
  - Deadline: Thursday Sept. 11
  - Turn in via file upload to ELMS/Canvas

Homework policies

- No public git forks of your homework!
  - Of course, a private git repo on github or elsewhere in the cloud is a good idea
- Adhere to the academic integrity policies of UMD/UMD CS

  http://osc.umd.edu/OSC/Default.aspx
  http://www.cs.umd.edu/class/resources/academicIntegrity.html
  - Discussing homework with colleagues is fine, but no sharing of code
  - No copying of code found online
  - No AI generated code

Next

- Acceleration structures for ray tracing
- Note: no class session on 9/4, instead
  - Read through slides independently
  - Ask questions to AI chatbot (Canvas Virtual Study Assistant, or other one)
  - Submit assignment (most interesting question and anwser from AI, in your opinion) on Canvas