3D Graphics

3D Graphics

3D graphics used in

- Games
- Animated movies
- Speciel effects in normal movies
- Training simulators (flying, sailing, military,. . .)
- Visualizations (data sets, architecture, . . .)

3D Graphics

3D graphics used in

- Games
- Animated movies
- Speciel effects in normal movies
- Training simulators (flying, sailing, military,. . .)
- Visualizations (data sets, architecture, . . .)

Same principles behind 3D graphics in all these settings.

Virtual Objects

Objects defined in mathematical 3D space ($\mathbb{R}^3$).

Virtual Objects

Objects defined in mathematical 3D space ($\mathbb{R}^3$).

We see surfaces of objects $\Rightarrow$ define surfaces.

Virtual Objects

Objects defined in mathematical 3D space ($\mathbb{R}^3$).

We see surfaces of objects $\Rightarrow$ define surfaces.

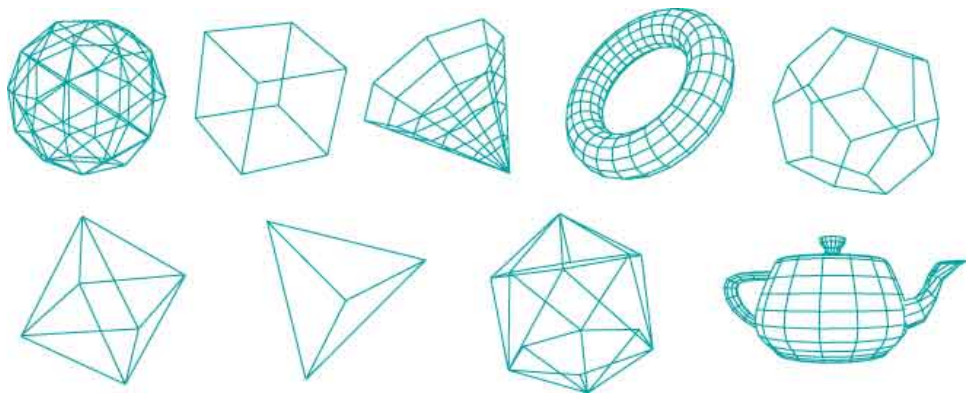
Triangles will be the fundamental element.

Virtual Objects

Objects defined in mathematical 3D space ($\mathbb{R}^3$).

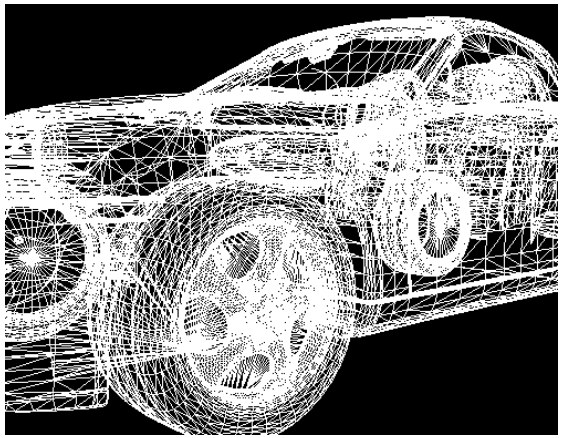
We see surfaces of objects $\Rightarrow$ define surfaces.

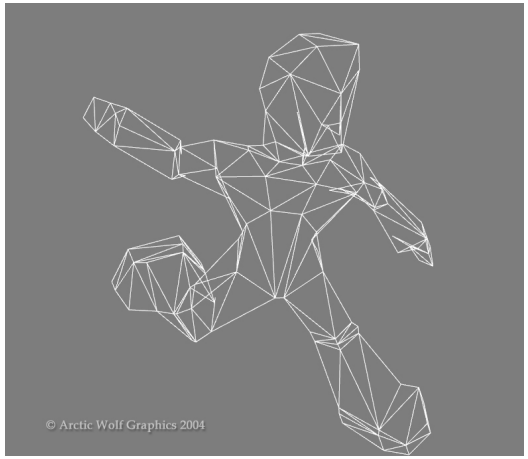
Triangles will be the fundamental element.

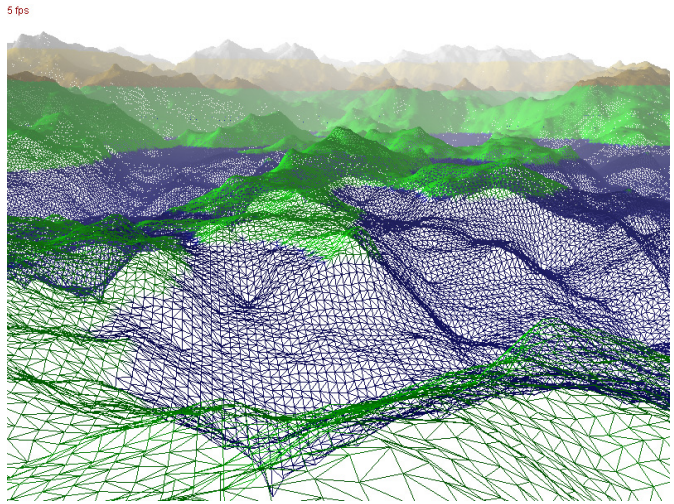


Virtual Objects

Triangles will be the fundamental element.







Rendering of 3D Virtual Objects

Main objective:

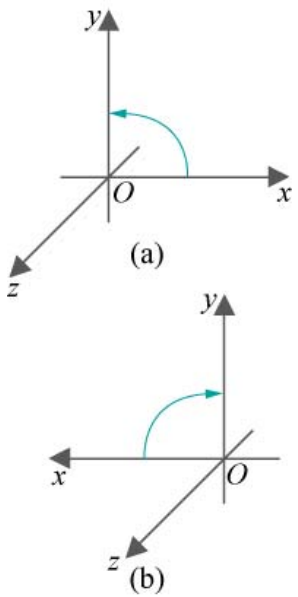
- Define models made out of triangles.
- Move models around in 3D space (*transformations*).
- Transfer to 2D screen space (*projection*).
- Add colors to the screen pixels covered by triangle (*shading*).

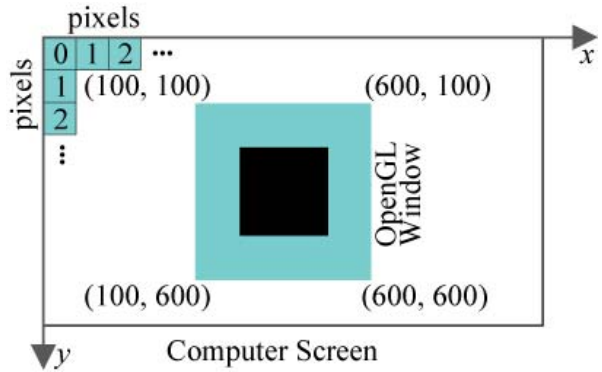
Rendering of 3D Virtual Objects

Main objective:

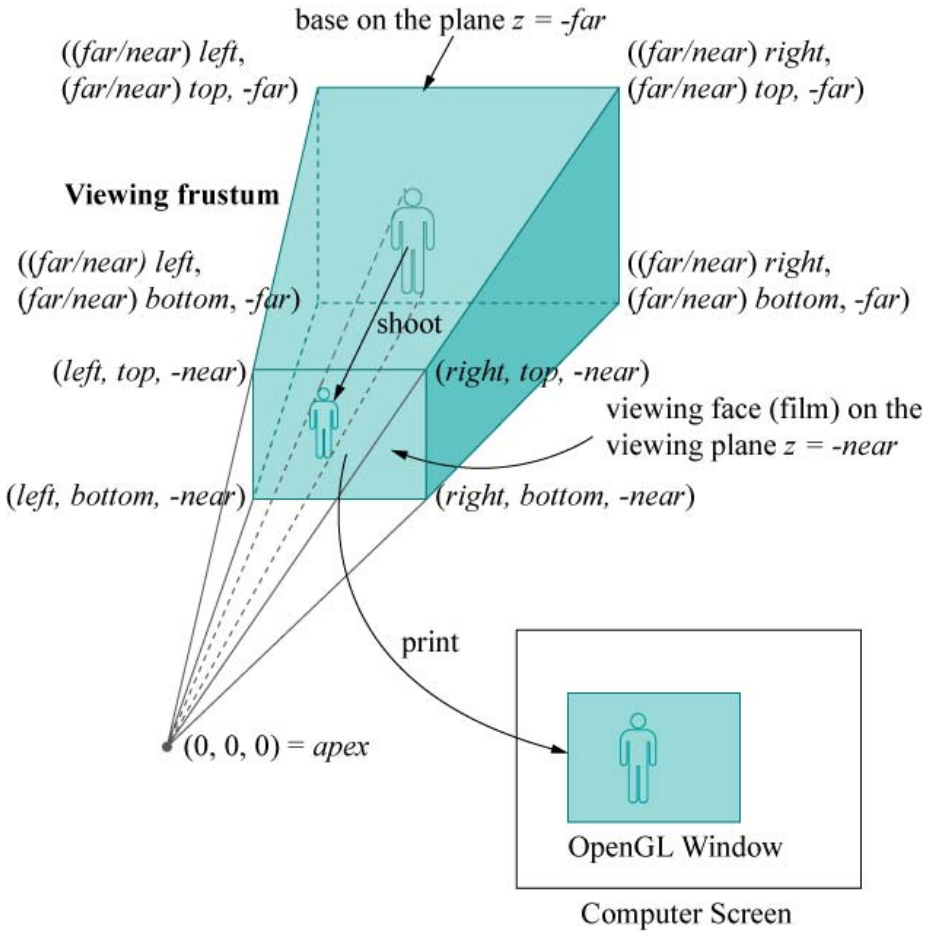
- Define models made out of triangles.
- Move models around in 3D space (*transformations*).
- Transfer to 2D screen space (*projection*).
- Add colors to the screen pixels covered by triangle (*shading*).

Coordinate systems:

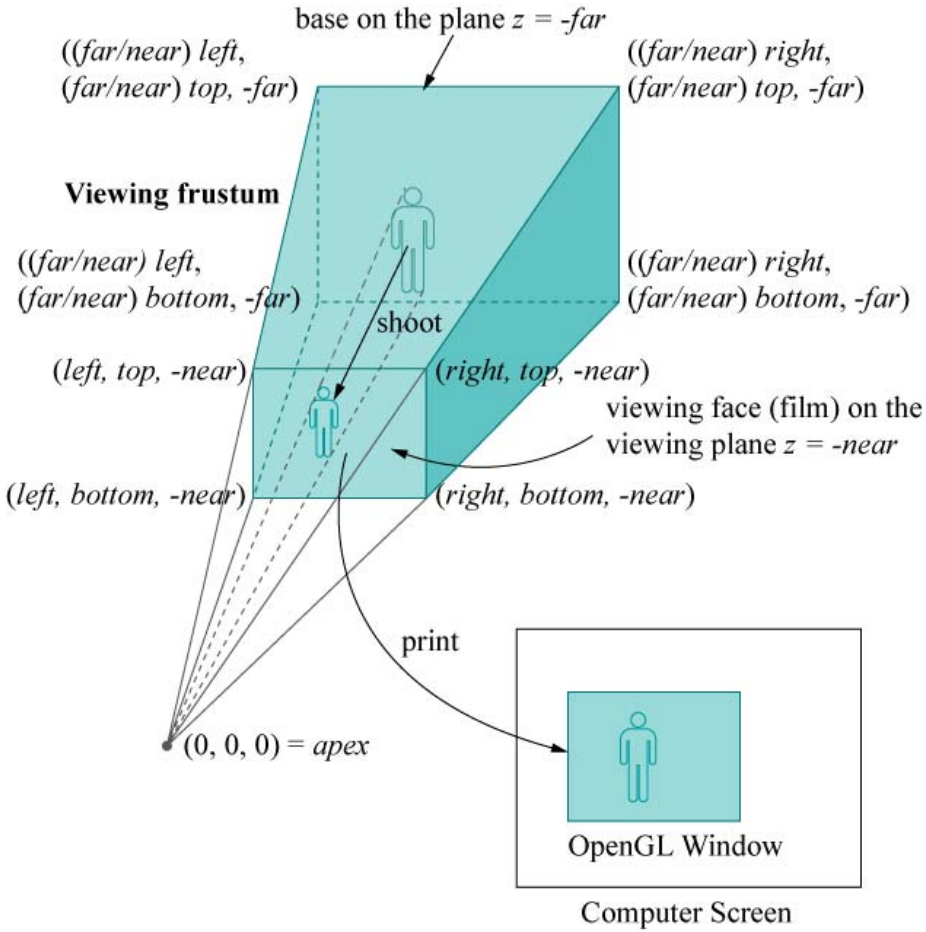


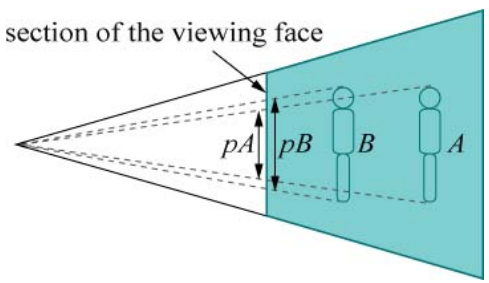


Perspective Projection



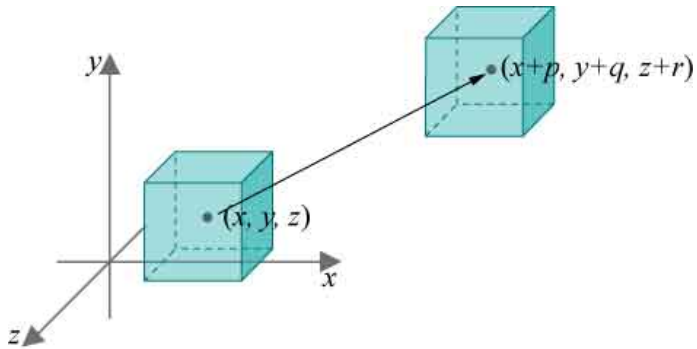
Perspective Projection





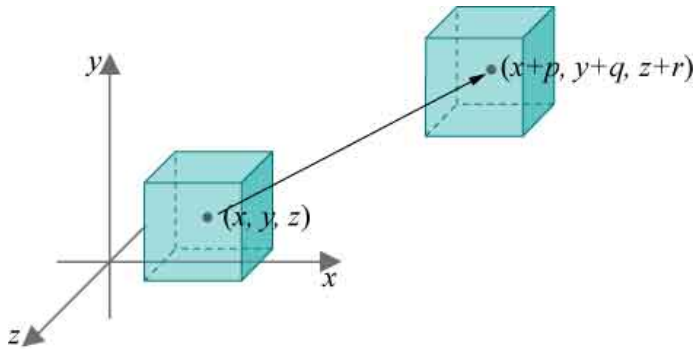
Moving Objects

We need to move our objects in 3D space.



Moving Objects

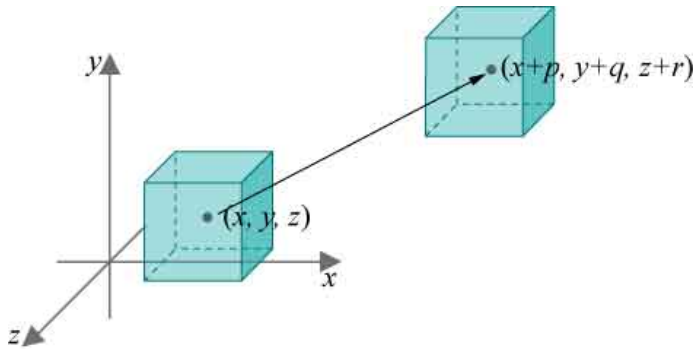
We need to move our objects in 3D space.



- An object/model (box, car, building, character,. . .) is defined in one position (often at coordinate system center). Will be needed in another position in the scene.

Moving Objects

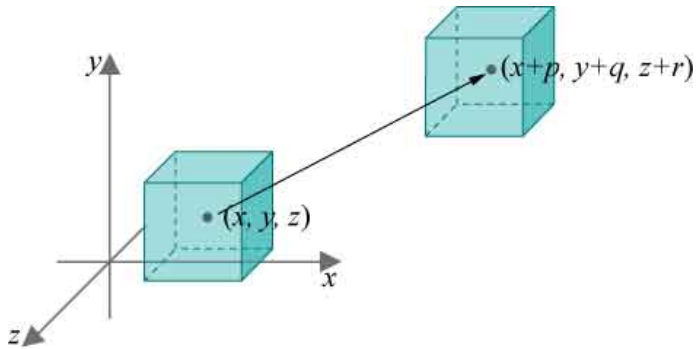
We need to move our objects in 3D space.



- An object/model (box, car, building, character,. . .) is defined in one position (often at coordinate system center). Will be needed in another position in the scene.
- Maybe in several places in one scene (town with houses and cars).

Moving Objects

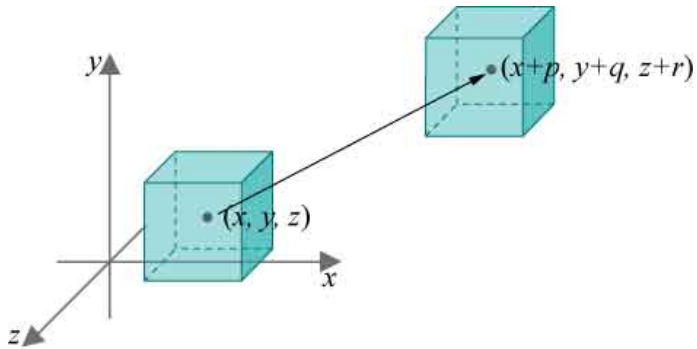
We need to move our objects in 3D space.



- An object/model (box, car, building, character,. . .) is defined in one position (often at coordinate system center). Will be needed in another position in the scene.
- Maybe in several places in one scene (town with houses and cars).
- Maybe in different places in different scenes/frames (animation).

Moving Objects

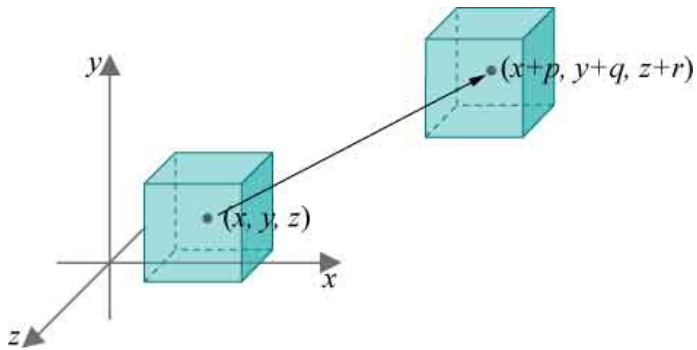
We need to move our objects in 3D space.



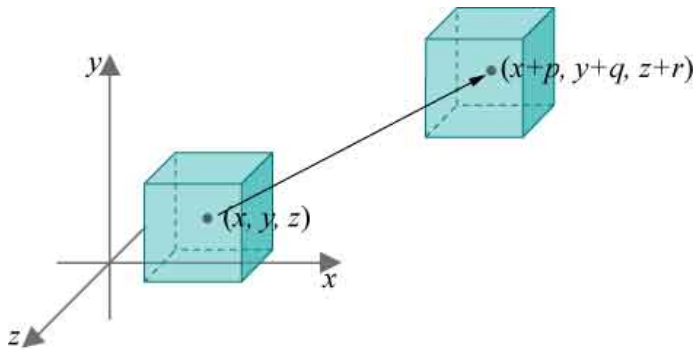
- An object/model (box, car, building, character,. . .) is defined in one position (often at coordinate system center). Will be needed in another position in the scene.
- Maybe in several places in one scene (town with houses and cars).
- Maybe in different places in different scenes/frames (animation).

Move model $\Leftrightarrow$ move triangles $\Leftrightarrow$ move points (vertices) $\Leftrightarrow$ $f : \mathbb{R}^3 \to \mathbb{R}^3$

Translation

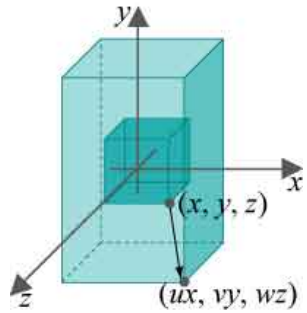


Translation

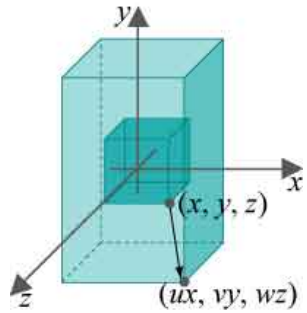


$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x+p \\ y+q \\ z+r \end{pmatrix}$$

Scaling

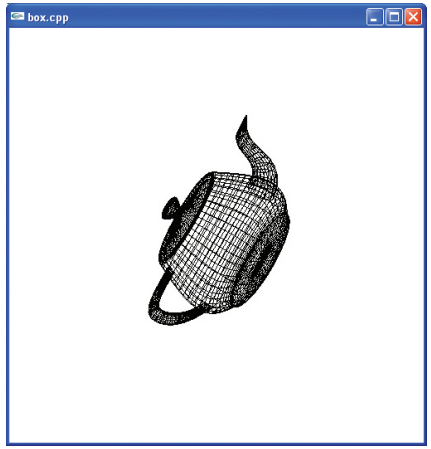


Scaling

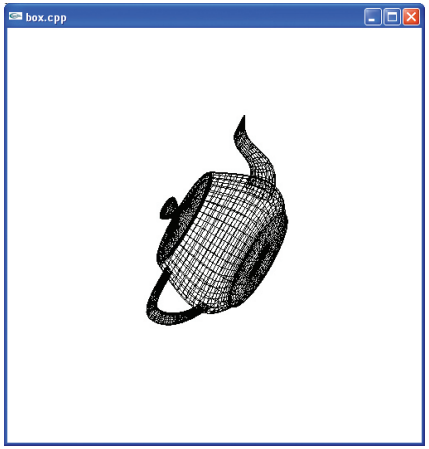


$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \cdot x \\ v \cdot y \\ w \cdot z \end{pmatrix}$$

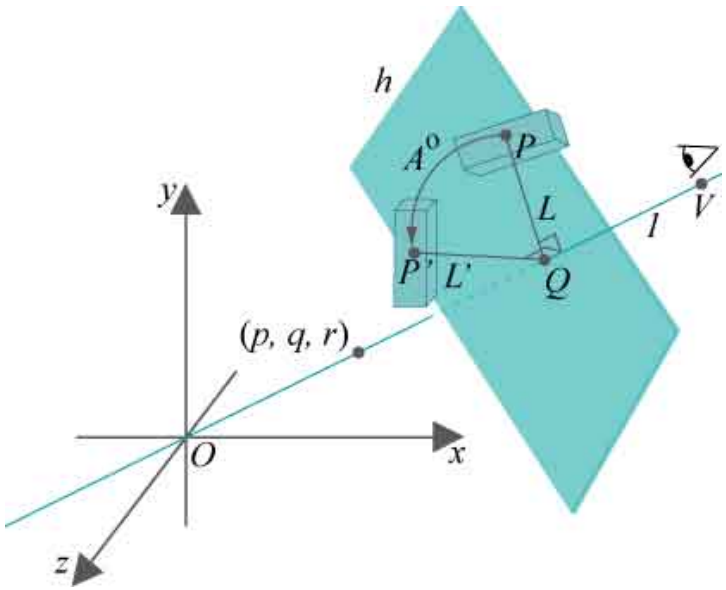
Rotation



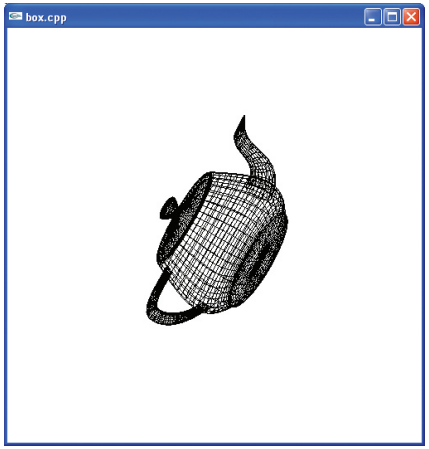
Rotation



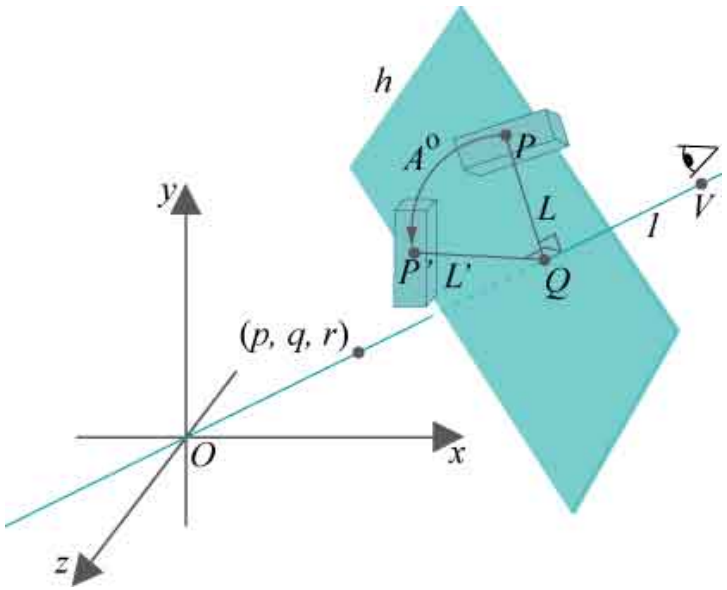
Rotation around line through origin:



Rotation



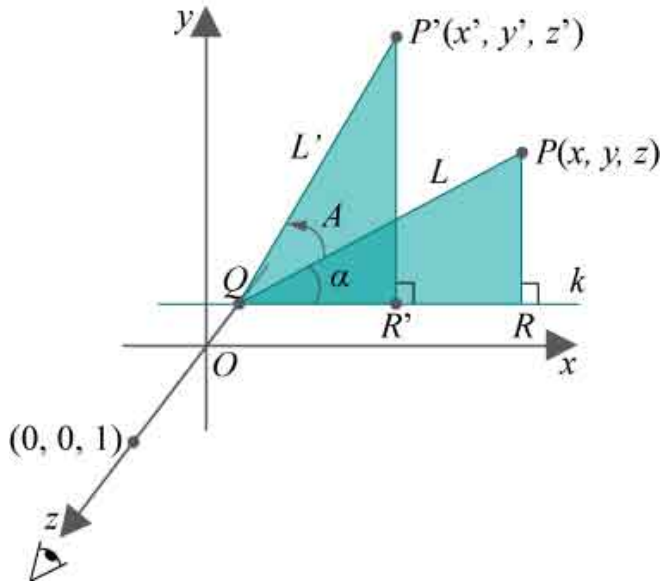
Rotation around line through origin:



$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix}$$

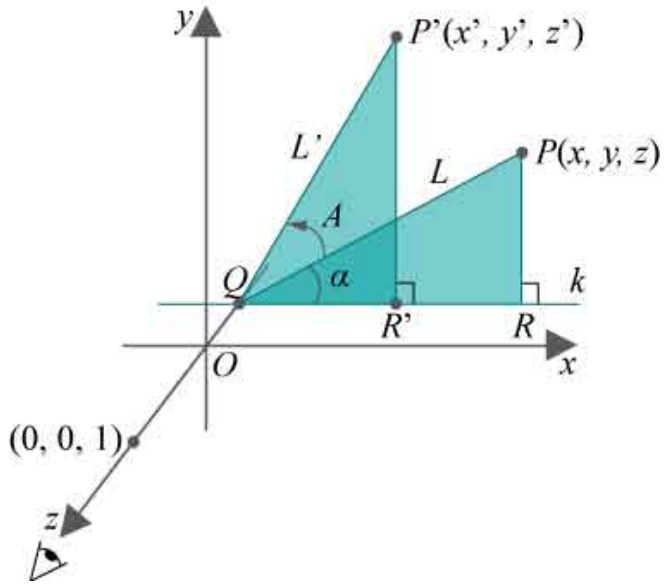
Rotation

Simpler case: Rotation around z-axis.



Rotation

Simpler case: Rotation around z-axis.



From formula for rotation in 2D (known from high school):

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \\ z \end{pmatrix}$$

Rotation

Similar: Rotation around x-axis and y-axis.

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y\cos\phi - z\sin\phi \\ y\sin\phi + z\cos\phi \end{pmatrix}$$

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} z\sin\phi + x\cos\phi \\ y \\ z\cos\phi - x\sin\phi \end{pmatrix}$$

Matrices

Move model $\Leftrightarrow$ move triangles $\Leftrightarrow$ move points (vertices) $\Leftrightarrow$ $f : \mathbb{R}^3 \to \mathbb{R}^3$

Matrices

Move model $\Leftrightarrow$ move triangles $\Leftrightarrow$ move points (vertices) $\Leftrightarrow f : \mathbb{R}^3 \to \mathbb{R}^3$

Any matrix induces a (linear) funktion $f : \mathbb{R}^3 \to \mathbb{R}^3$:

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1x + 2y + 3z \\ 4x + 5y + 6z \\ 7x + 8y + 9z \end{pmatrix}$$

Matrices

Move model $\Leftrightarrow$ move triangles $\Leftrightarrow$ move points (vertices) $\Leftrightarrow f : \mathbb{R}^3 \to \mathbb{R}^3$

Any matrix induces a (linear) funktion $f : \mathbb{R}^3 \to \mathbb{R}^3$:

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1x + 2y + 3z \\ 4x + 5y + 6z \\ 7x + 8y + 9z \end{pmatrix}$$

Matrix multiplication is associative: $A \cdot (B \cdot C) = (A \cdot B) \cdot C$.

Matrices

Move model $\Leftrightarrow$ move triangles $\Leftrightarrow$ move points (vertices) $\Leftrightarrow f : \mathbb{R}^3 \to \mathbb{R}^3$

Any matrix induces a (linear) funktion $f : \mathbb{R}^3 \to \mathbb{R}^3$:

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1x + 2y + 3z \\ 4x + 5y + 6z \\ 7x + 8y + 9z \end{pmatrix}$$

Matrix multiplication is associative: $A \cdot (B \cdot C) = (A \cdot B) \cdot C$. Hence:

$$A \cdot (B \cdot (C \cdot (E \cdot (F \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix})))) = ((((A \cdot B) \cdot C) \cdot E) \cdot F) \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Matrices

Move model $\Leftrightarrow$ move triangles $\Leftrightarrow$ move points (vertices) $\Leftrightarrow f : \mathbb{R}^3 \to \mathbb{R}^3$

Any matrix induces a (linear) funktion $f : \mathbb{R}^3 \to \mathbb{R}^3$:

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1x + 2y + 3z \\ 4x + 5y + 6z \\ 7x + 8y + 9z \end{pmatrix}$$

Matrix multiplication is associative: $A \cdot (B \cdot C) = (A \cdot B) \cdot C$. Hence:

$$A \cdot (B \cdot (C \cdot (E \cdot (F \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix})))) = ((((A \cdot B) \cdot C) \cdot E) \cdot F) \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Saves calculations: 3D object = many triangles = many points. All points go through the same sequence of transformations (moves). Calculate the matrix product once.

Matrices

Move model $\Leftrightarrow$ move triangles $\Leftrightarrow$ move points (vertices) $\Leftrightarrow f : \mathbb{R}^3 \to \mathbb{R}^3$

Any matrix induces a (linear) funktion $f : \mathbb{R}^3 \to \mathbb{R}^3$:

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1x + 2y + 3z \\ 4x + 5y + 6z \\ 7x + 8y + 9z \end{pmatrix}$$

Matrix multiplication is associative: $A \cdot (B \cdot C) = (A \cdot B) \cdot C$. Hence:

$$A \cdot (B \cdot (C \cdot (E \cdot (F \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix})))) = ((((A \cdot B) \cdot C) \cdot E) \cdot F) \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Saves calculations: 3D object = many triangles = many points. All points go through the same sequence of transformations (moves). Calculate the matrix product once.

Question: can all our needed transformations be expressed as matrices?

Transformations as Matrices

Transformations as Matrices

- Scaling

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \cdot x \\ v \cdot y \\ w \cdot z \end{pmatrix} = \begin{bmatrix} u & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & w \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Transformations as Matrices

- Scaling

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \cdot x \\ v \cdot y \\ w \cdot z \end{pmatrix} = \begin{bmatrix} u & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & w \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Rotation angle ϕ around the z-axis

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \\ z \end{pmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Transformations as Matrices

- Scaling

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \cdot x \\ v \cdot y \\ w \cdot z \end{pmatrix} = \begin{bmatrix} u & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & w \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Rotation angle ϕ around the z-axis

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \\ z \end{pmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Translation?

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x + p \\ y + q \\ z + r \end{pmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Transformations as Matrices

- Scaling

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u \cdot x \\ v \cdot y \\ w \cdot z \end{pmatrix} = \begin{bmatrix} u & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & w \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Rotation angle ϕ around the z-axis

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \\ z \end{pmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Translation?

$$f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x + p \\ y + q \\ z + r \end{pmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

No. For translation we have $f(\vec{0}) \neq \vec{0})$, but all functions given by matrices take $\vec{0}$ to $\vec{0}$.

Homogeneous Coordinates

Go to 4D:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Homogeneous Coordinates

Go to 4D:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

And back:

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \rightarrow \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

Homogeneous Coordinates

Translations (in 3D) can now be expressed as matrix multiplication:

$$\begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+p \\ y+q \\ z+r \\ 1 \end{pmatrix}$$

Homogeneous Coordinates

Translations (in 3D) can now be expressed as matrix multiplication:

$$\begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+p \\ y+q \\ z+r \\ 1 \end{pmatrix}$$

All 3x3 matrices are still available (incl. scaling and rotation):

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 1x+2y+3z \\ 4x+5y+6z \\ 7x+8y+9z \\ 1 \end{pmatrix}$$

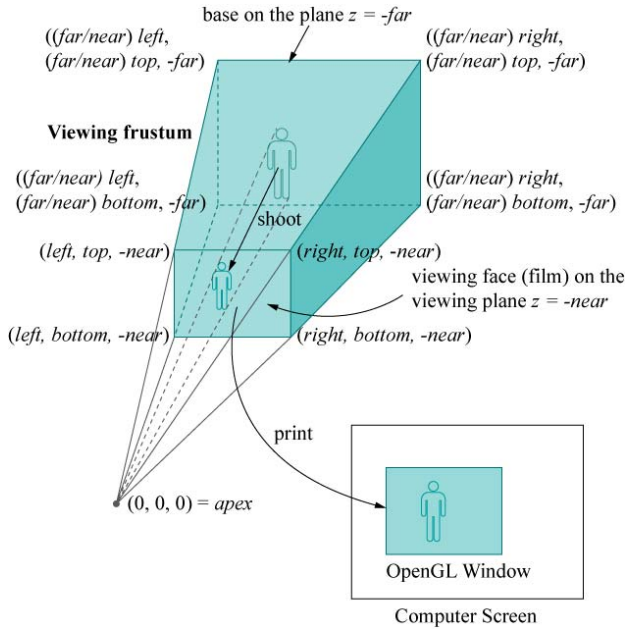
Projection

Projection to screen: $f : \mathbb{R}^3 \to \mathbb{R}^2$.

Projection

Projection to screen: $f : \mathbb{R}^3 \to \mathbb{R}^2$.

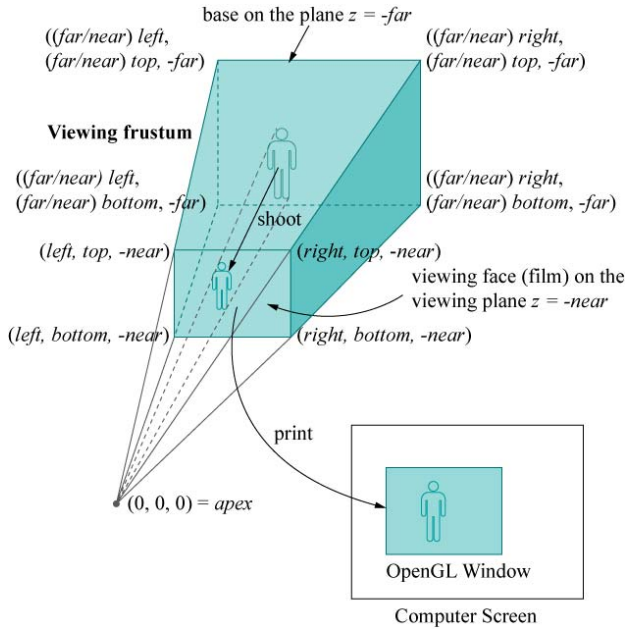
Prespective projection:



Projection

Projection to screen: $f : \mathbb{R}^3 \to \mathbb{R}^2$.

Prespective projection:



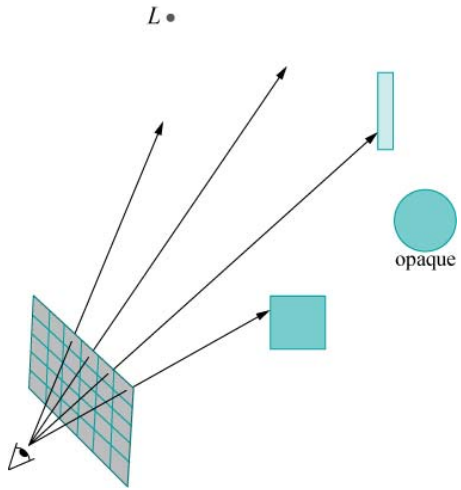
Expressed as 4x4 matrix multiplication ($d = -\text{near}$):

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \to \begin{pmatrix} xd/z \\ yd/z \\ d \end{pmatrix}$$

Shading

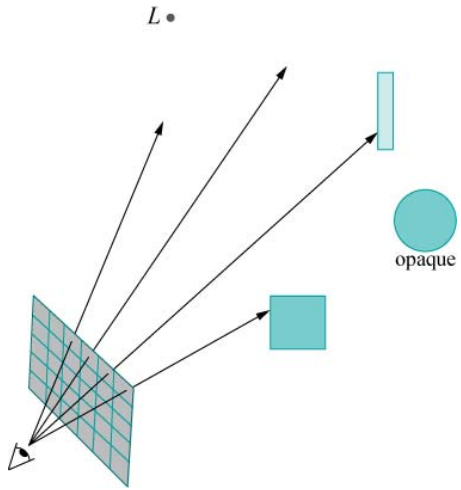
Shading

Shading = find color values at pixels of screen (when rendering a virtual 3D scene).



Shading

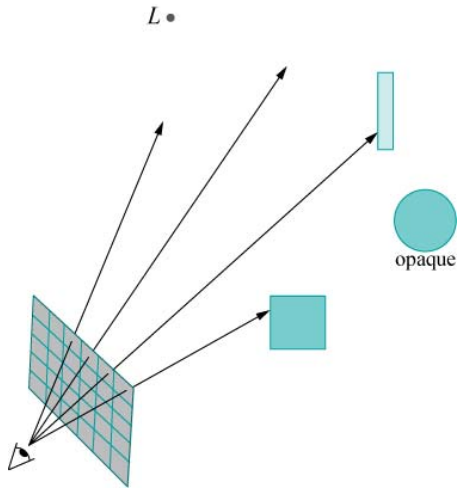
Shading = find color values at pixels of screen (when rendering a virtual 3D scene).



Same as finding color value for the closest triangle on the ray of the pixel (assuming this is an opaque object, and air is clear).

Shading

Shading = find color values at pixels of screen (when rendering a virtual 3D scene).



Same as finding color value for the closest triangle on the ray of the pixel (assuming this is an opaque object, and air is clear).

Core objective: Find color values for intersection of a ray with a triangle.

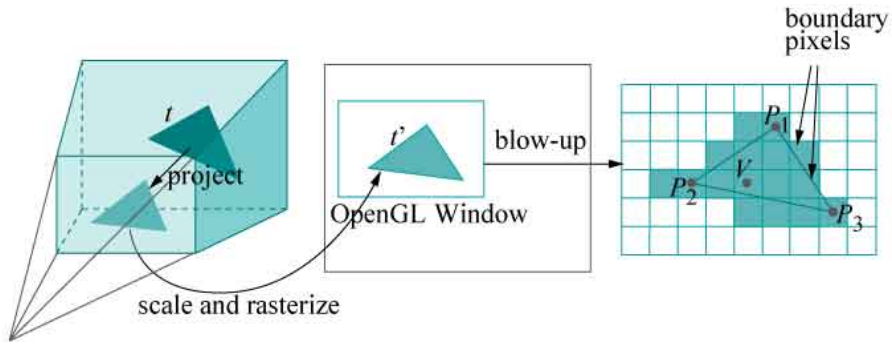
Shading

Core objective: Find color values for point at intersection of a ray with a triangle.

Shading

Core objective: Find color values for point at intersection of a ray with a triangle.

- Rendering is triangle-driven (`foreach triangle: render`).
- Triangles are simply (triples of) vertices until rasterization phase, where pixels of the triangle are found from pixels of the vertices.



So the actual rays are determined in the rasterization phase.

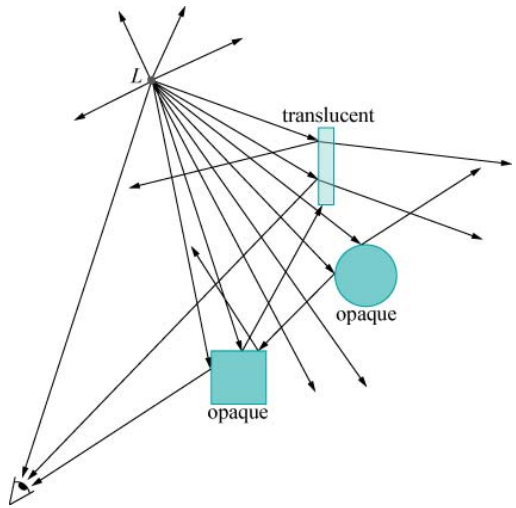
Modeling Light

Core objective: Find color values for intersection of a ray with a triangle.

Modeling Light

Core objective: Find color values for intersection of a ray with a triangle.

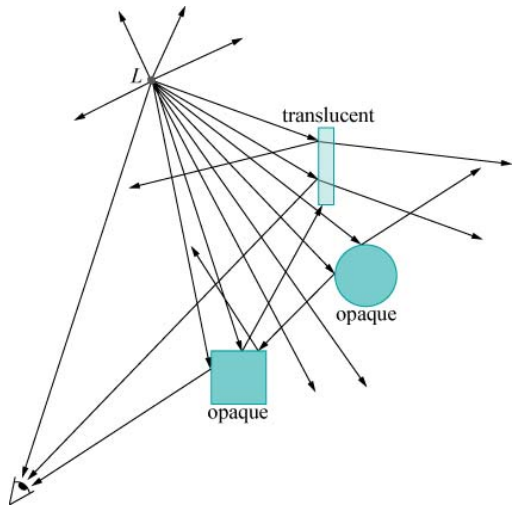
Model physical light (photons)



Modeling Light

Core objective: Find color values for intersection of a ray with a triangle.

Model physical light (photons)



Photons are

- Emitted from ligth sources.
- Reflected, absorbed, re-emitted, transmitted when hitting objects.

Modeling Light

Highly complex physical proces. Zillions of photons.

Can only be modeled to a certain degree mathematically (ongoing research expands on the available models).

Modeling Light

Highly complex physical proces. Zillions of photons.

Can only be modeled to a certain degree mathematically (ongoing research expands on the available models).

(Figure by Jason Jacobs)

Modeling Light

Realtime rendering additionally has severe time constraints. Framerate $\sim$ 30/sec, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Modeling Light

Realtime rendering additionally has severe time constraints. Framerate $\sim$ 30/sec, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Hence, *realtime* rendering (games, simulators) use quite rough light models.

Modeling Light

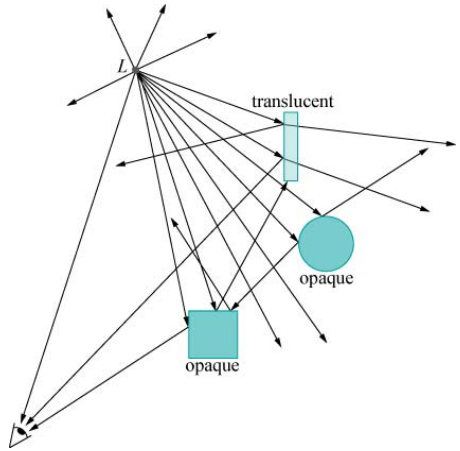
Realtime rendering additionally has severe time constraints. Framerate $\sim$ 30/sec, screen size $\sim 10^6$ pixels $\Rightarrow$ few GPU cycles available for calculation per ray.

Hence, *realtime* rendering (games, simulators) use quite rough light models.

Offline rendering (movies, visualization) can use more advanced light models (and also other rendering methods needing more time, such as ray tracing and radiosity).

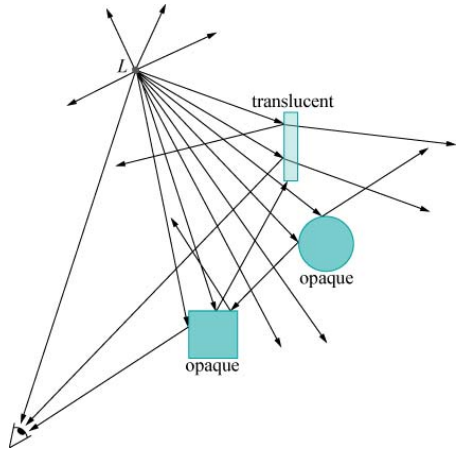
Phongs Lightning Model

A classic, simple model.



Phongs Lightning Model

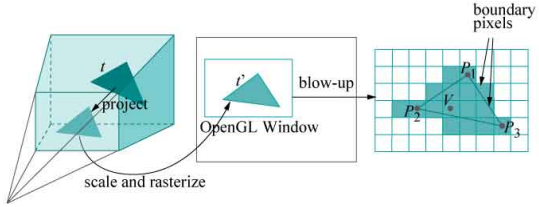
A classic, simple model.



- Models only opaque objects.
- Models only one level of light/surface interactions.
- Light/surface interaction is modeled by two simple submodels, diffuse and specular term.
- Models indirect light effects very crudely (ambient term).
- Light actually generated at surface can be added (emissive term).
- Occlusion is not modeled (all objects see all lights).

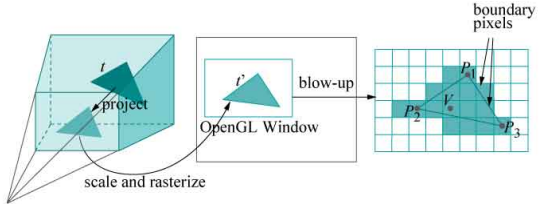
Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?



Shading models

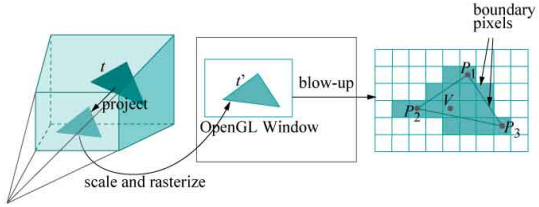
So we have information in each vertex. How spread color calculation over entire triangle pixels?



- Flat shading: Color calculated for one point is used for entire triangle.

Shading models

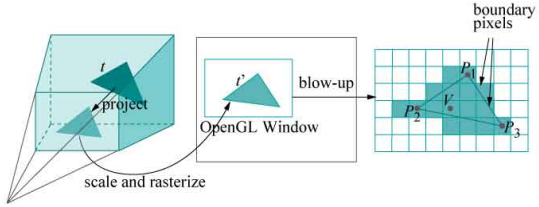
So we have information in each vertex. How spread color calculation over entire triangle pixels?



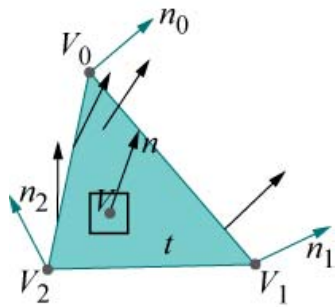
- Flat shading: Color calculated for one point is used for entire triangle.
- Smooth shading (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).

Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?

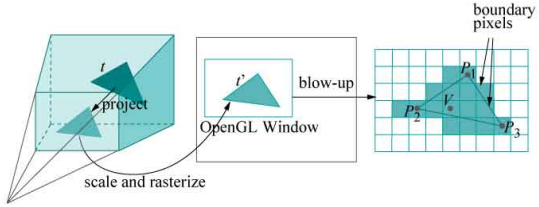


- Flat shading: Color calculated for one point is used for entire triangle.
- Smooth shading (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).
- Phong shading: Color calculation done for all points of pixels.

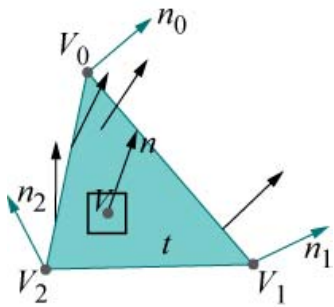


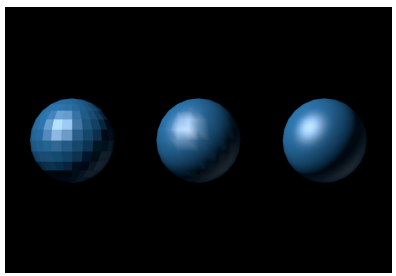
Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?



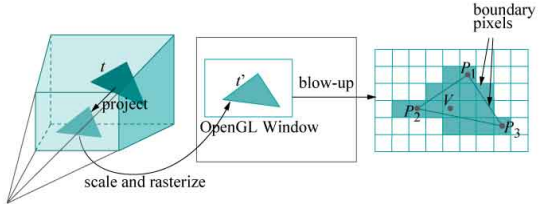
- Flat shading: Color calculated for one point is used for entire triangle.
- Smooth shading (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).
- Phong shading: Color calculation done for all points of pixels.



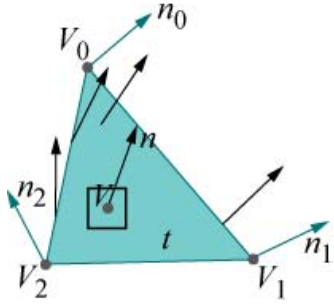


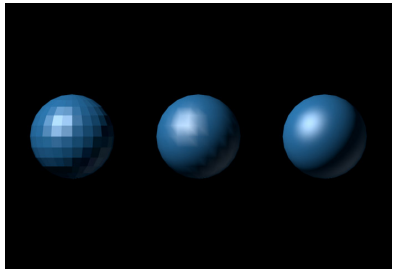
Shading models

So we have information in each vertex. How spread color calculation over entire triangle pixels?



- Flat shading: Color calculated for one point is used for entire triangle.
- Smooth shading (aka. Gouraud shading): Colors calculated for three vertices are interpolated across the entire triangle (individually for each RGB-channel).
- Phong shading: Color calculation done for all points of pixels.





Calculation time increases down the list.

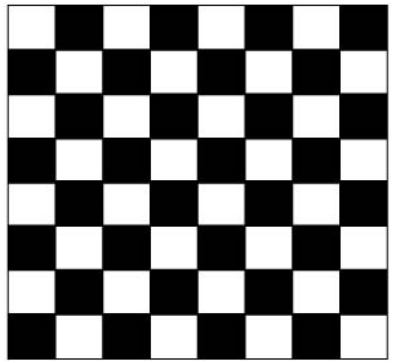
Textures

Texture = 1/2/3D data table.

Textures

Texture = 1/2/3D data table.

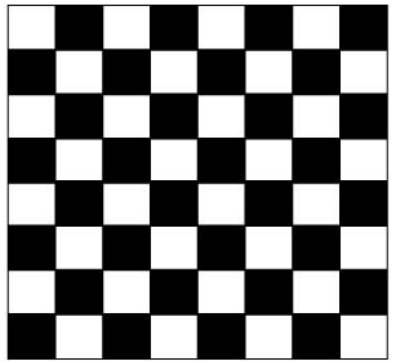
Often: (color values of) 2D picture.



Textures

Texture = 1/2/3D data table.

Often: (color values of) 2D picture.

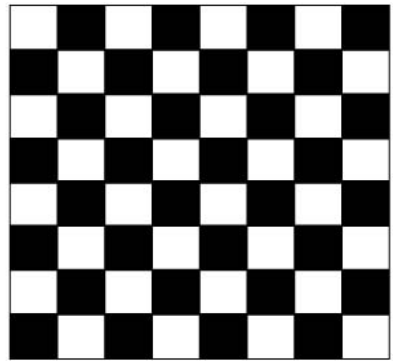


External file, or generated online inside program (animated textures), or renderede (offline or online) scene.

Textures

Texture = 1/2/3D data table.

Often: (color values of) 2D picture.



External file, or generated online inside program (animated textures), or renderede (offline or online) scene.

But texture data can be interpreted as anything, e.g. normal vectors, light maps/shadow maps, heightfields,. . . .

Use of Textures

- Generate detailed graphical content hardly possible with triangles (such as clouds, skyboxes, plain pictures (posters, decals) on surfaces).
- Create illusion of structure, saving lots of triangles. Can be (low level) part of a level-of-detail scheme.
- Most of a game's graphical expression is via artwork using textures.
- Hold special-purpose data for use in rendering process.

Examples

(From All Things Designed)

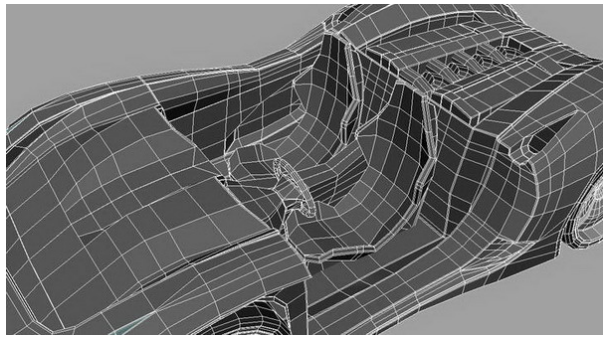
Examples

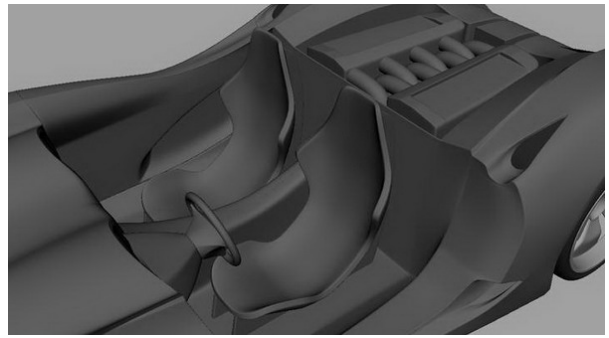
Examples

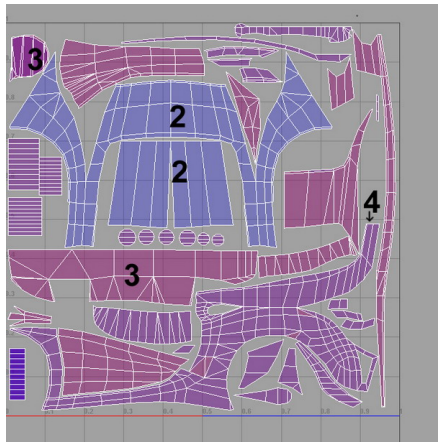
Examples

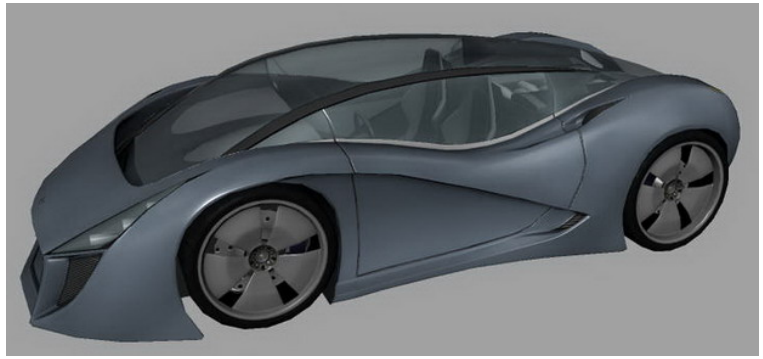
(From Sly Cooper)

Examples



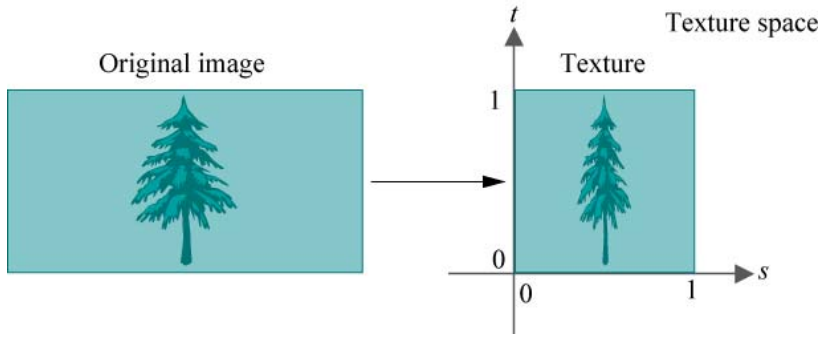






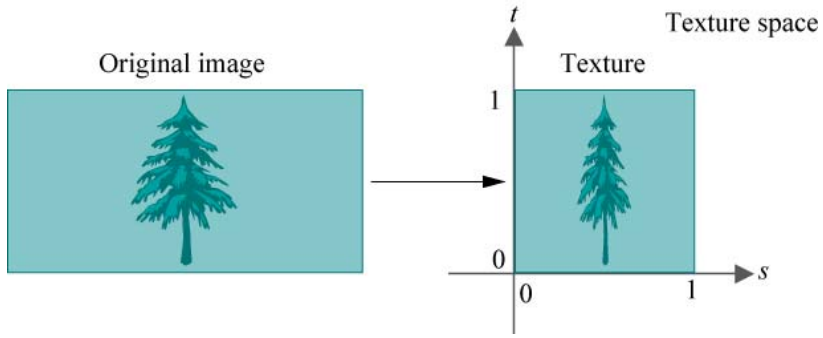
(Figures by Valentin Nadolu)

Texture Coordinates



Texture data get mapped to $[0; 1]^{1,2,3}$ in texture space.

Texture Coordinates

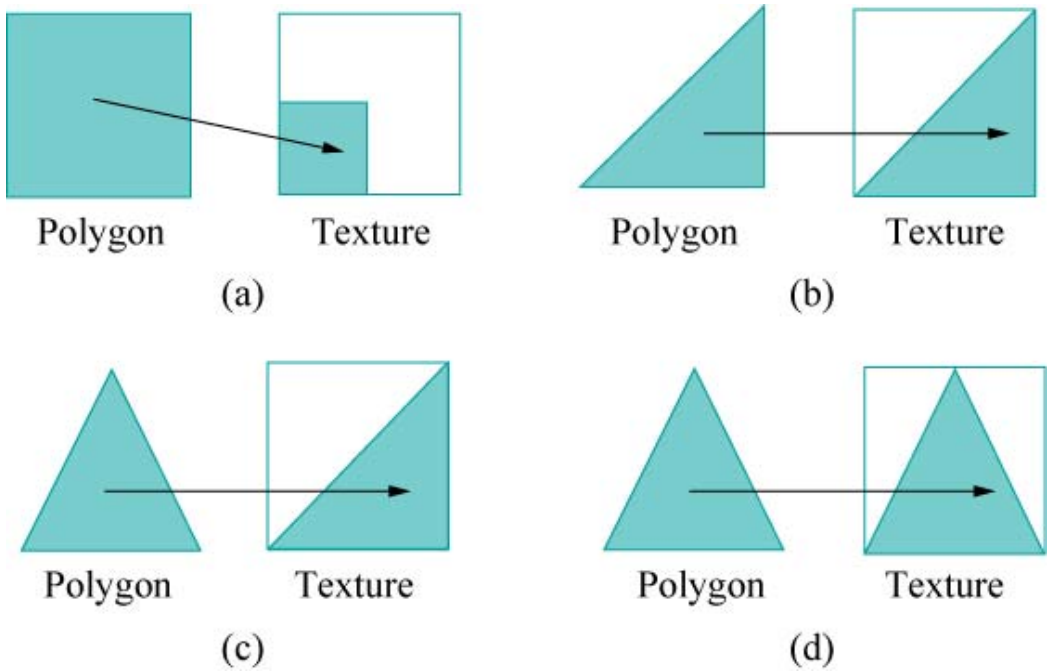


Texture data get mapped to $[0; 1]^{1,2,3}$ in texture space.

Vertices can be associated with texture coordinates

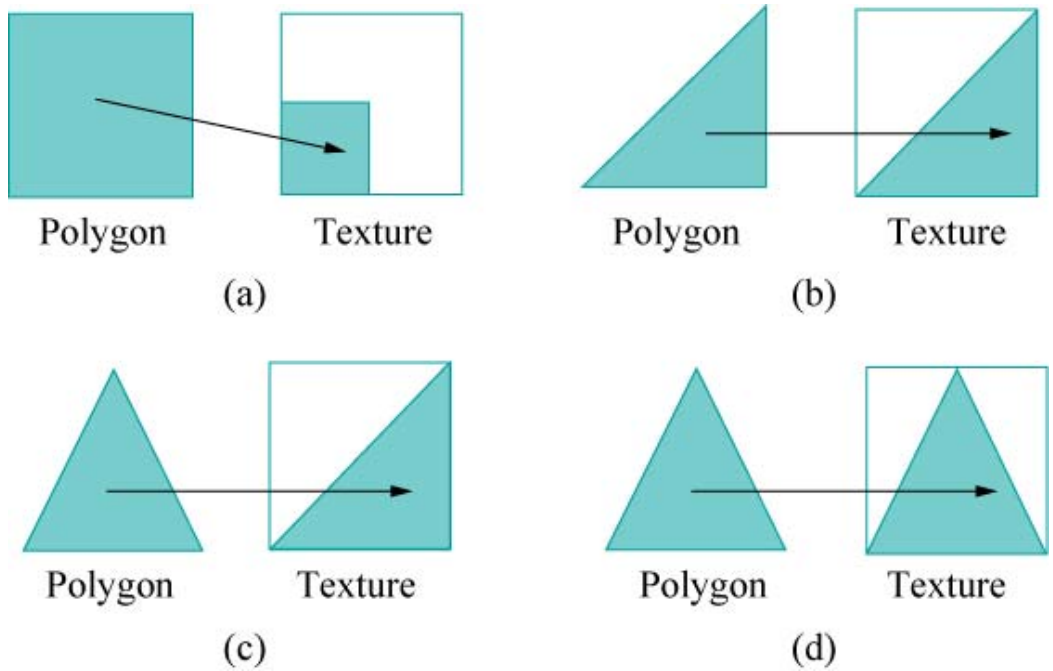
Texture Coordinates

Texture space points can be arbitrary:



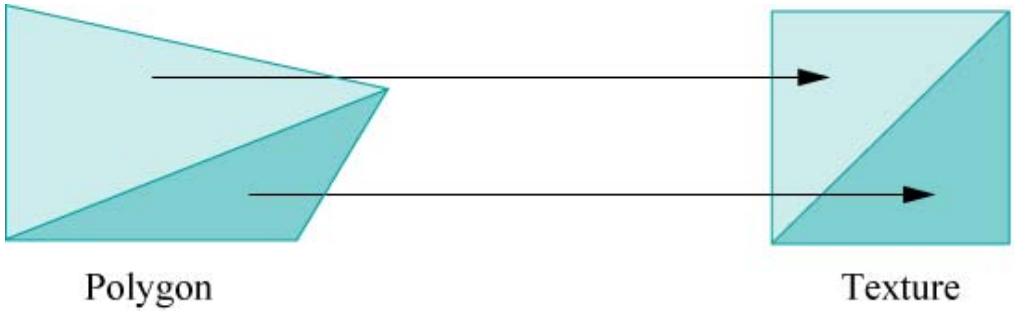
Texture Coordinates

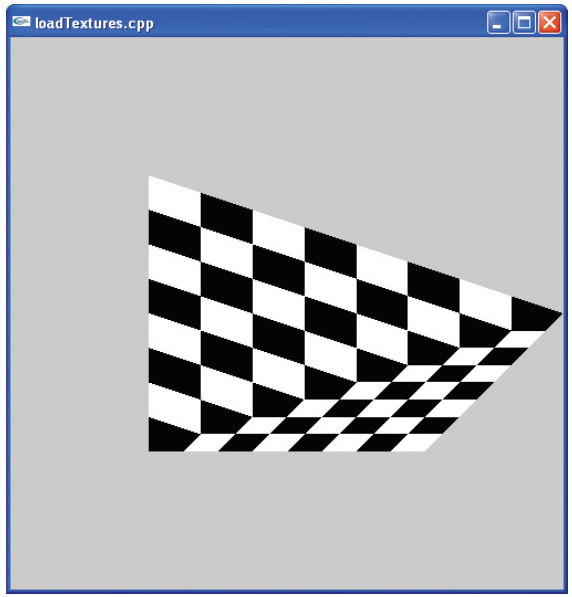
Texture space points can be arbitrary:



Points internally in triangle are associated with points in texture space using interpolation.

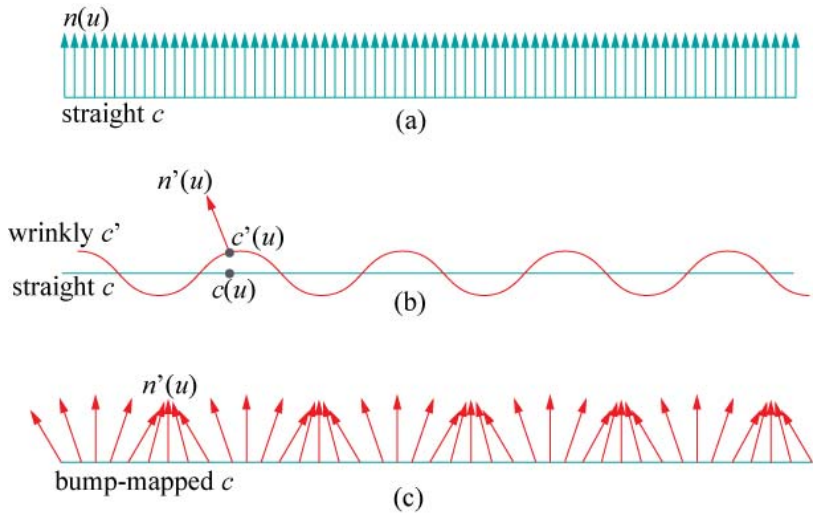
Interpolation Example





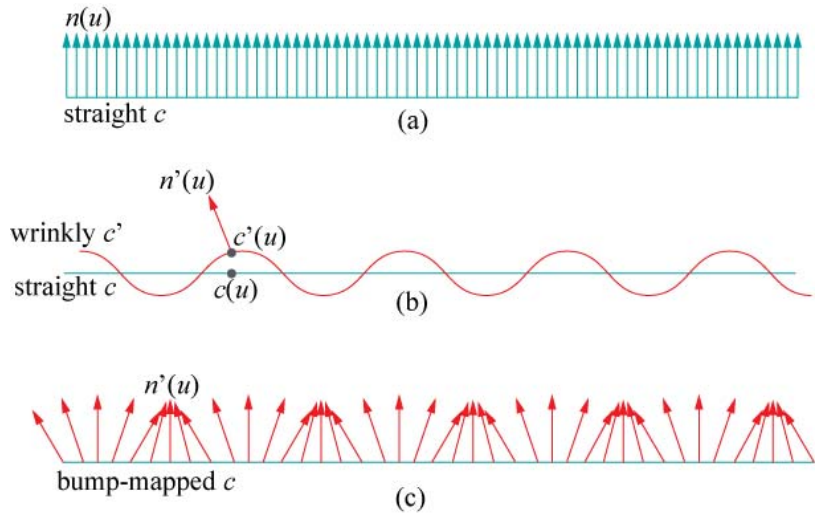
Texture Use: Bumpmapping

Store surface normals (or perturbation of normals) in texture.



Texture Use: Bumpmapping

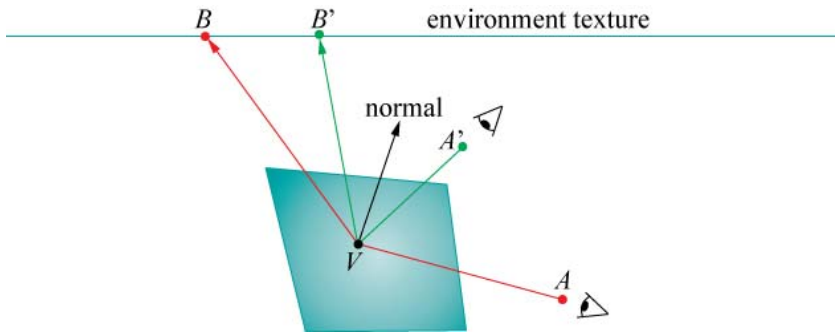
Store surface normals (or perturbation of normals) in texture.



(Figure by `www.chromesphere.com`)

Texture Use: Environment Mapping

Relections can see environment. Make part of shading calculation.



Environment Mapping

Easiest with Cube mapping:

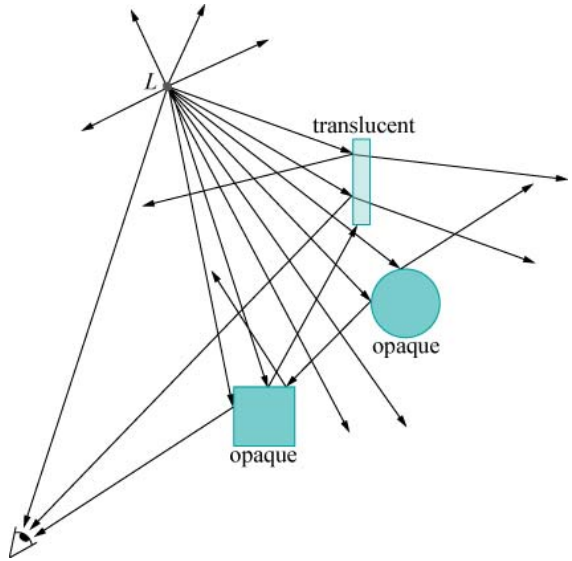
Six single textures. Can each be generated online by rendering from current center and saving framebuffer as texture.

Alternative Rendering Methods

- Standard GPU pipeline (OpenGL): real-time, but shading based on *local* effects. No shadows in basic pipeline (must be added by ad-hoc methods).
- Ray tracing: *Global* shading model particularly good at specular effects (shiny surfaces). Too computationally expensive to be real-time.
- Radiosity: *Global* shading model particularly good at diffuse effects (matte surfaces, indirect light). Too computationally expensive to be real-time. But well suited for storing results as textures (as diffuse light is not viewpoint dependent).

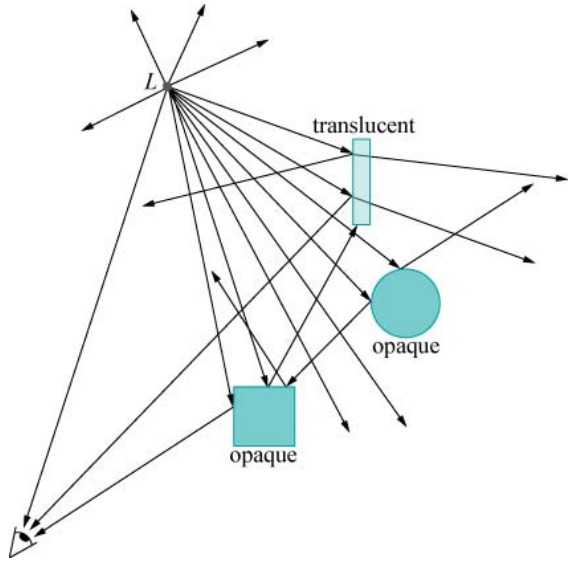
Ray Tracing

Follow photon paths to the eye.



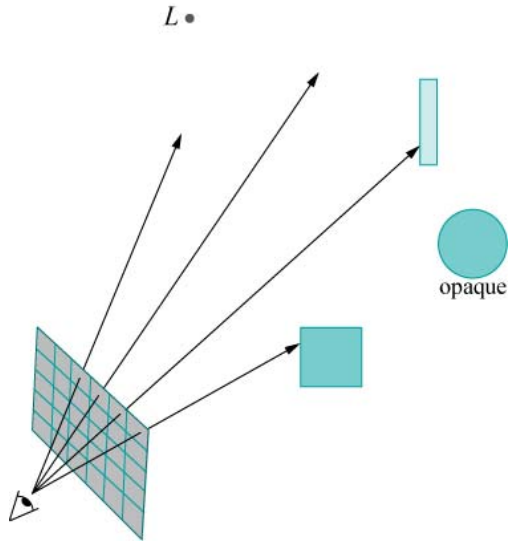
Ray Tracing

Follow photon paths to the eye.



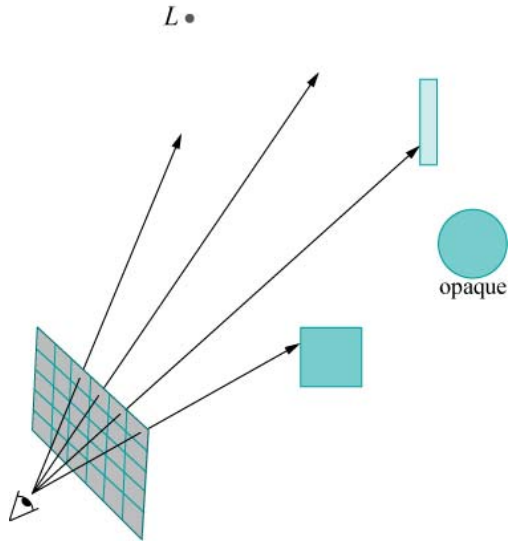
For efficiency, follow these in a backwards fashion, from the eye (only spend time on photons actually hitting the eye).

Ray Tracing Level 0



At end of rays: calculate colors by Phongs lighting model.

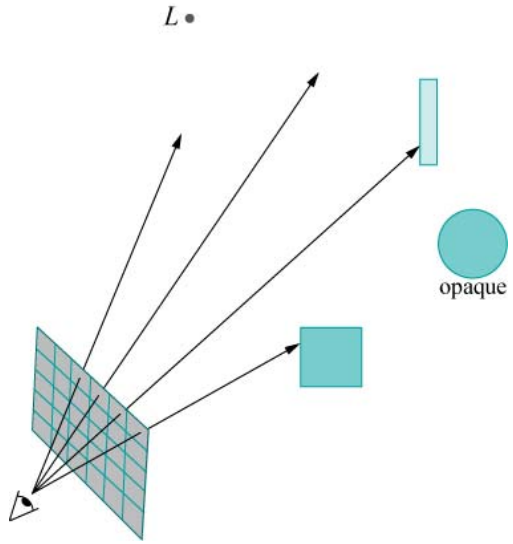
Ray Tracing Level 0



At end of rays: calculate colors by Phongs lighting model.

Same result as standard GPU pipeline.

Ray Tracing Level 0



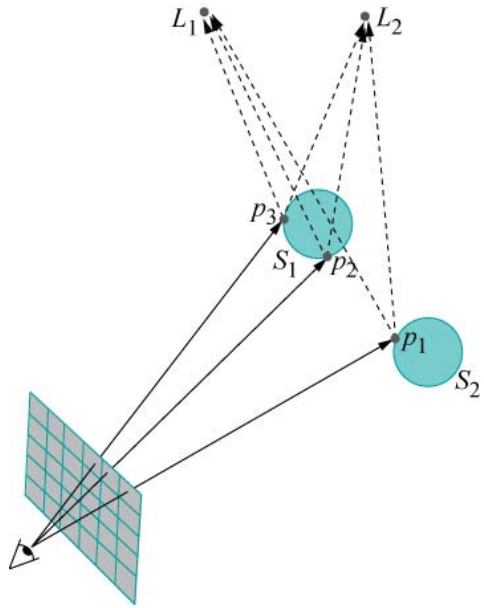
At end of rays: calculate colors by Phongs lighting model.

Same result as standard GPU pipeline.

Requires mechanism for fast determination of intersection points between rays and objects of the scene (e.g., store objects in data structures).

Ray Tracing Level 1

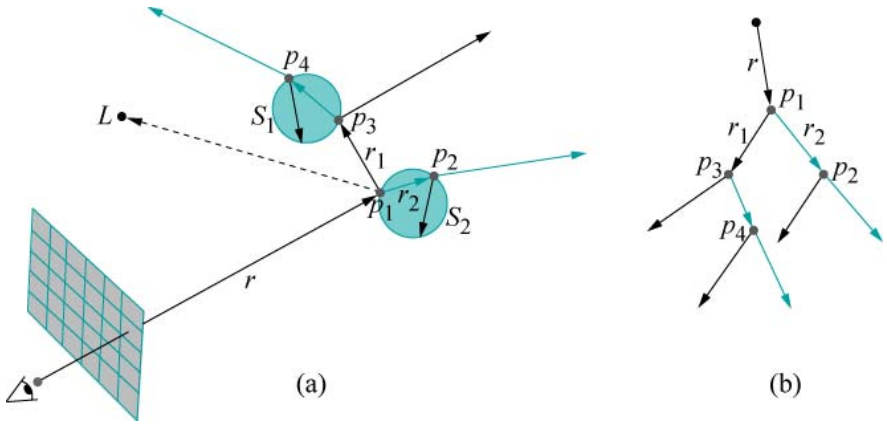
Add occlusion tests to light sources.



Gives shadows.

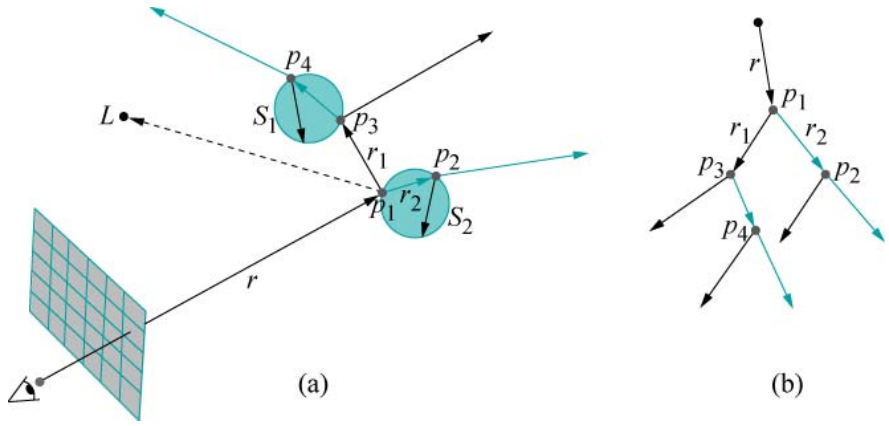
Ray Tracing Level 2+

Add reflection and transmission. Then *recurse*.



Ray Tracing Level 2+

Add reflection and transmission. Then *recurse*.



Note: simulating indirect light transfer between diffuse surfaces requires following many (approximating infinitely many) reflective rays from each ray intersection point in the recursive process.

Prohibitively costly. So ray tracing works best for glossy materials.

Ray Tracing Examples

(Figures by Bill Martin, RayScale, Daniel Pohl, NVIDIA)

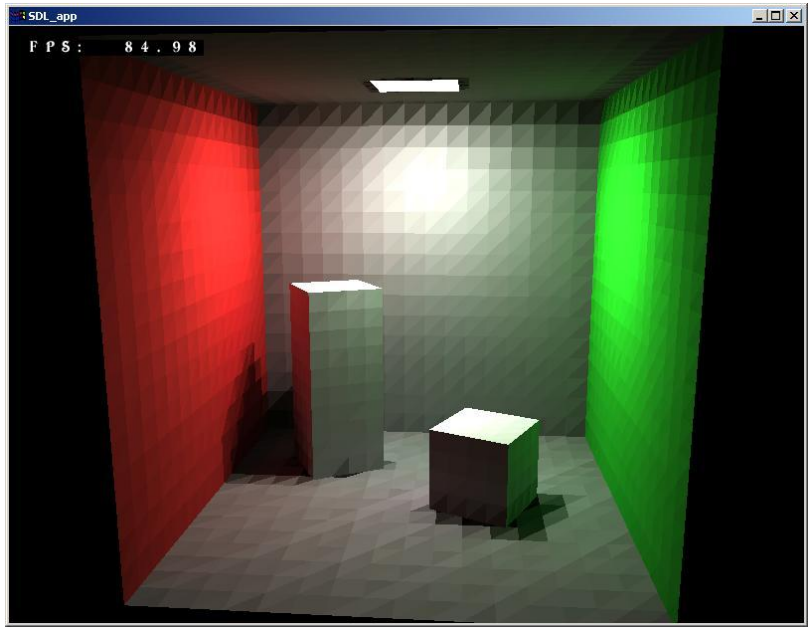
Radiosity

Model indirect light bouncing between purely diffuse (Lambertian) surfaces (of which some are light emitting).

(Figure by Jason Jacobs)

Patches

Start by patch-ifying the surfaces of the scene.



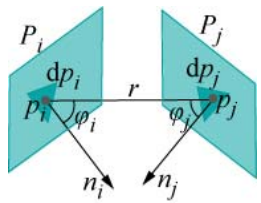
(Figure by Chuck Pheatt)

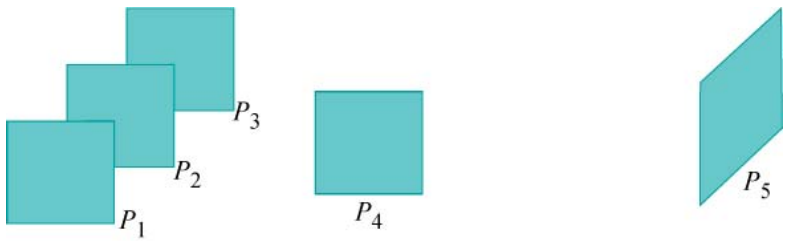
Entire patch will be considered to have same ligth value (radiosity/brightness) B_i.

Radiosity: photons emitted per time and per area.

Form Factors

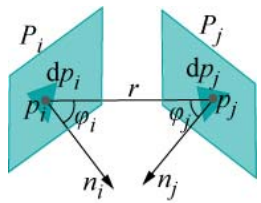
Form factor F_{ij}: measure of light transport between patch i and j.

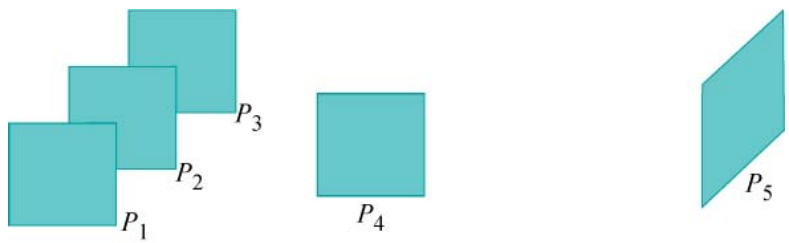




Form Factors

Form factor F_{ij}: measure of light transport between patch i and j.





(Technically: For F_{ij}: sum (integrate) contribution between (infinitesimal small areas around) all points on the two patches P_i and P_j.)

Radiosity Equation

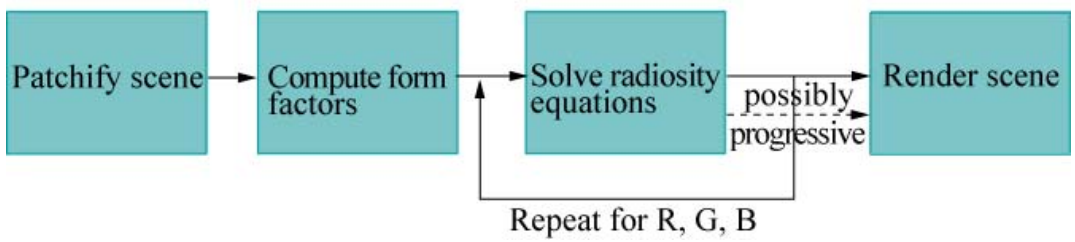
With M a specific $n \times n$ matrix (n is number of patches in schene) having entries depending on form factors and reflectance of patches, B the sought vector of brightness/radiosity values for patches and E the vector of emissive values for patches, one can prove:

$$MB = E$$

Using properties of the matrix M and results from matrix theory, it can be proven that the iterative process

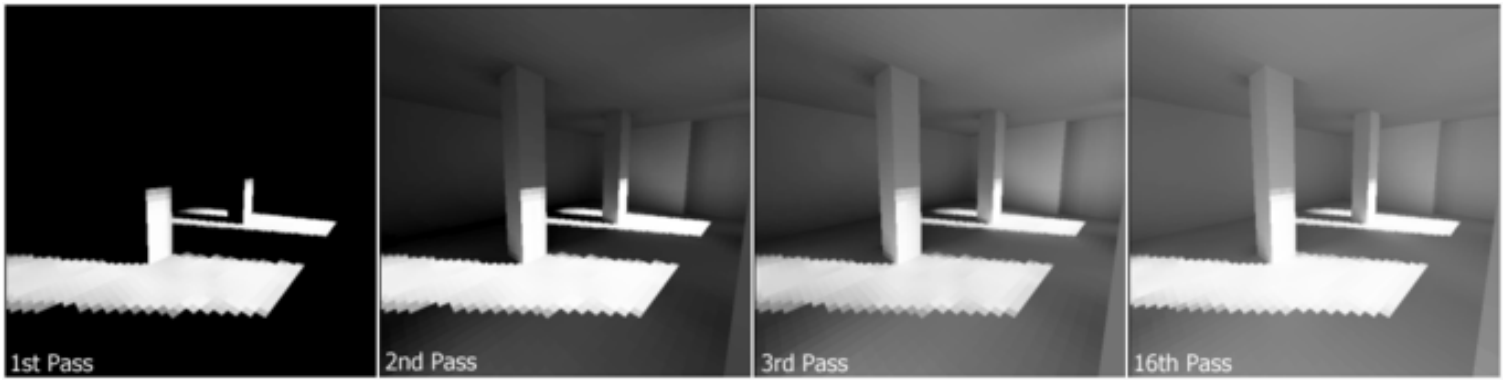
$$B_{i+1} = E + (I - M)B_i$$

for any start vector B_0 will converge to B. This is usually faster than directly solving $MB = E$ (by e.g. inverting M), and less memory is used.



Iterative Process

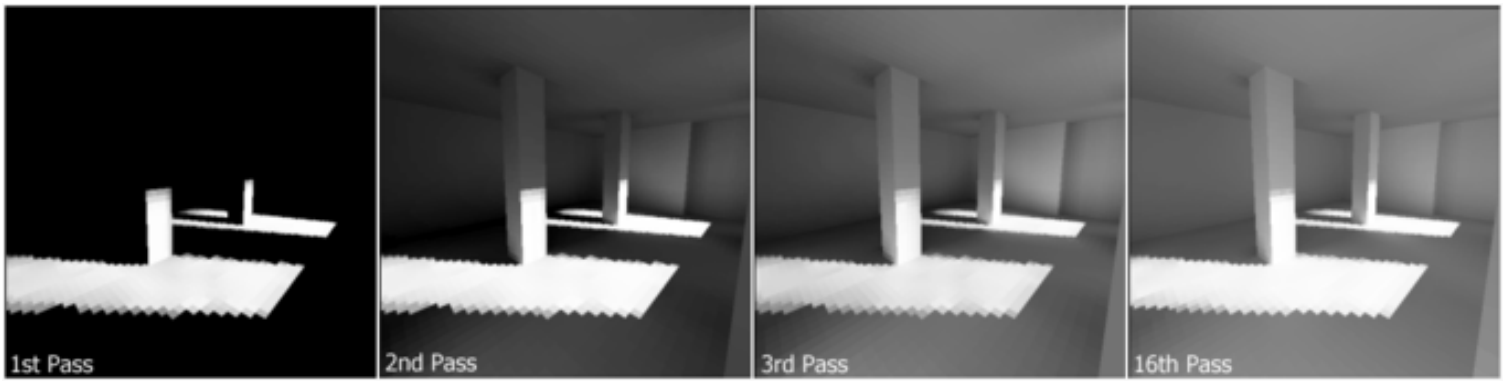
Here is the result of rendering a specific scene with B_1, B_2, B_3, B_{16}.



(Figure by Hugo Elias)

Iterative Process

Here is the result of rendering a specific scene with B_1, B_2, B_3, B_{16}.



(Figure by Hugo Elias)

The patching of the room may be refined based on one run of radiosity, increasing the resolution in areas with large variation in light values (edges of shadows, e.g.), and lowering the resolution in areas with small variation.