SM 14/15 – T8
Computer Graphics and Animation

LCC, MIERSI

Miguel Tavares Coimbra

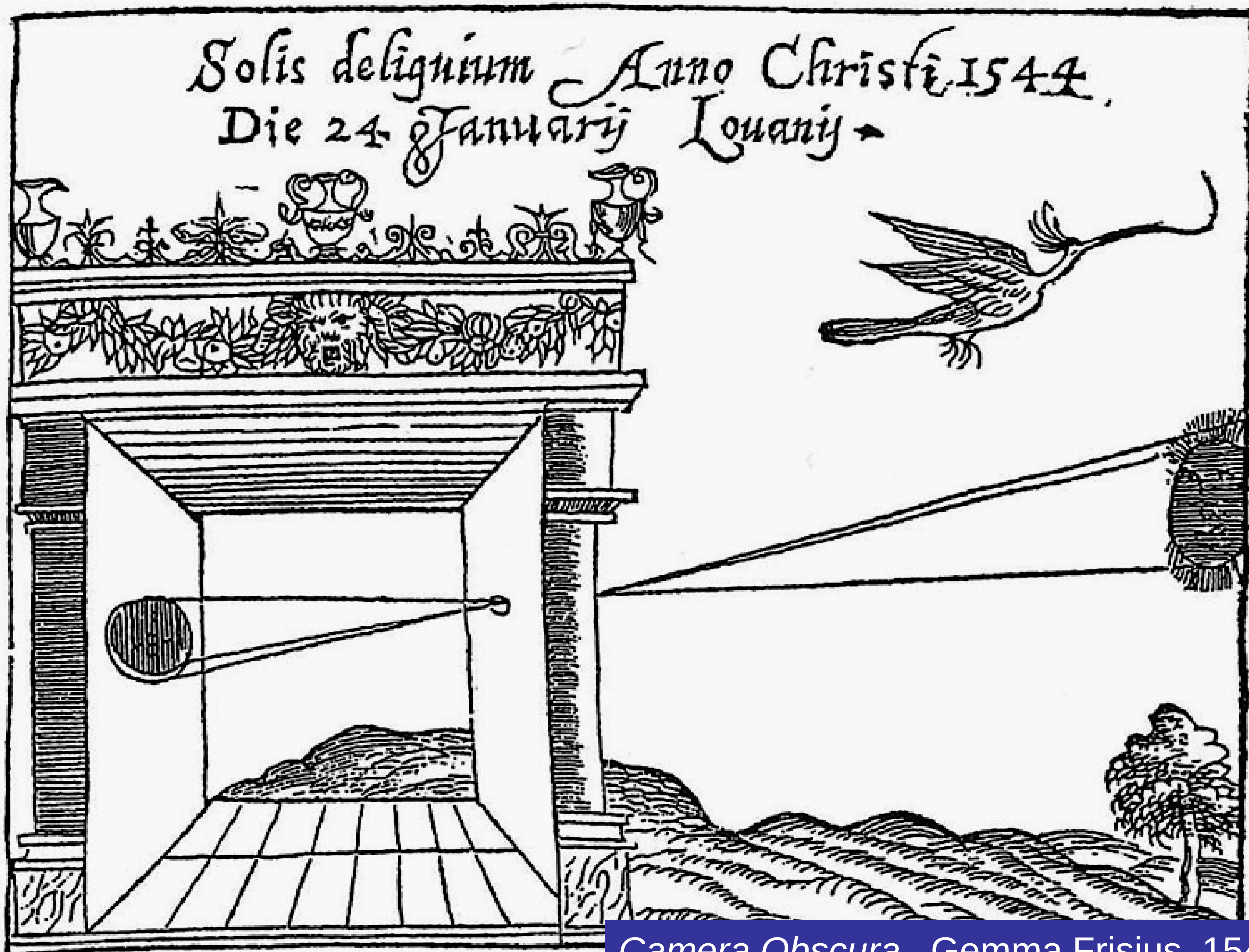
U. PORTO

(Some) Pieces of the Puzzle

- Image creators (3D -> 2D)
  - Camera (T4)
  - **Computer Graphics (Today)**
- Image manipulators (2D -> 2D)
  - Image Processing (T4)
- Image displays (2D -> ?)
  - 2D Screen
  - 3D Virtual Reality (T7)

How do we get 2D images of a real 3D world?

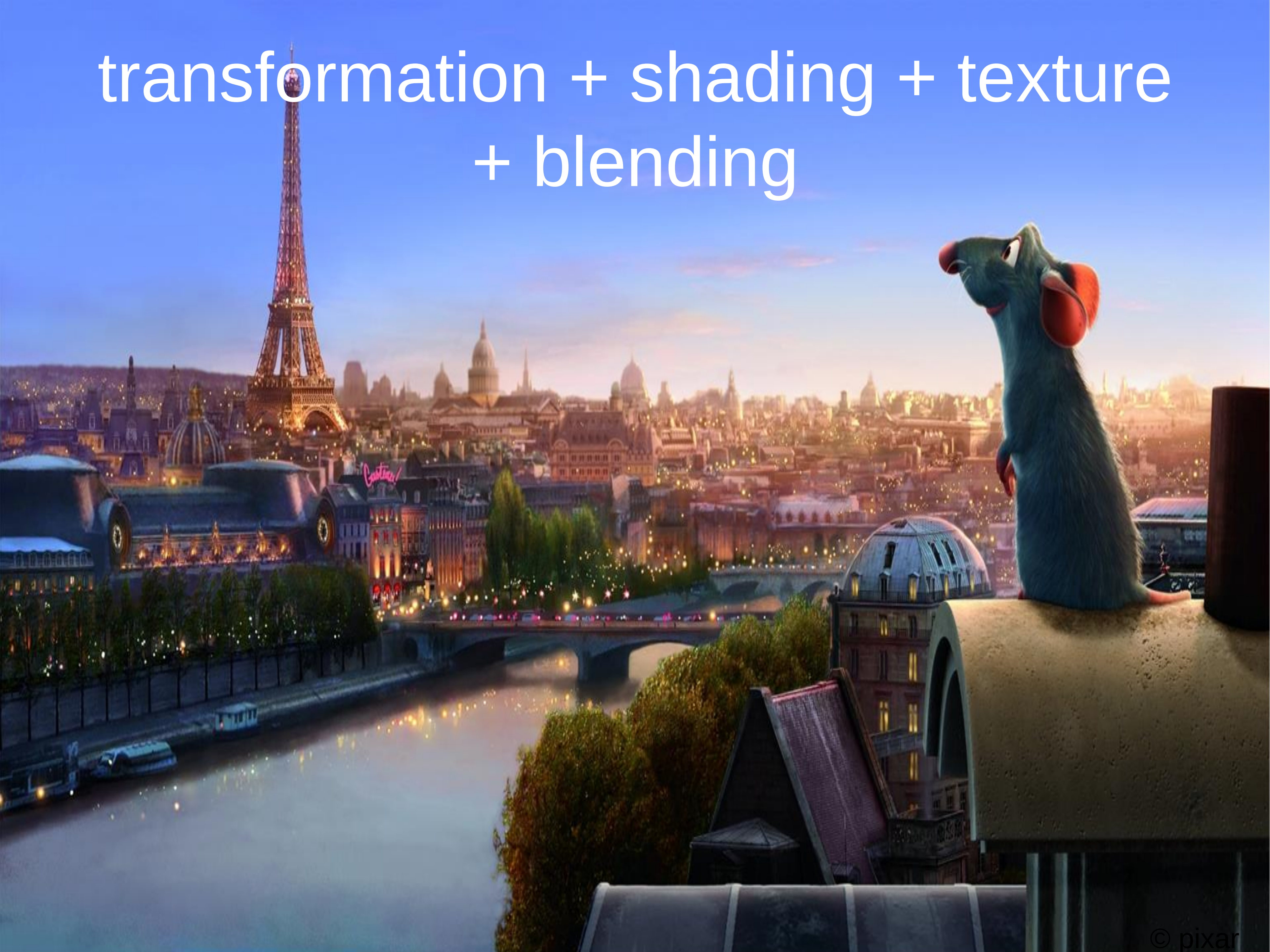
U.PORTO



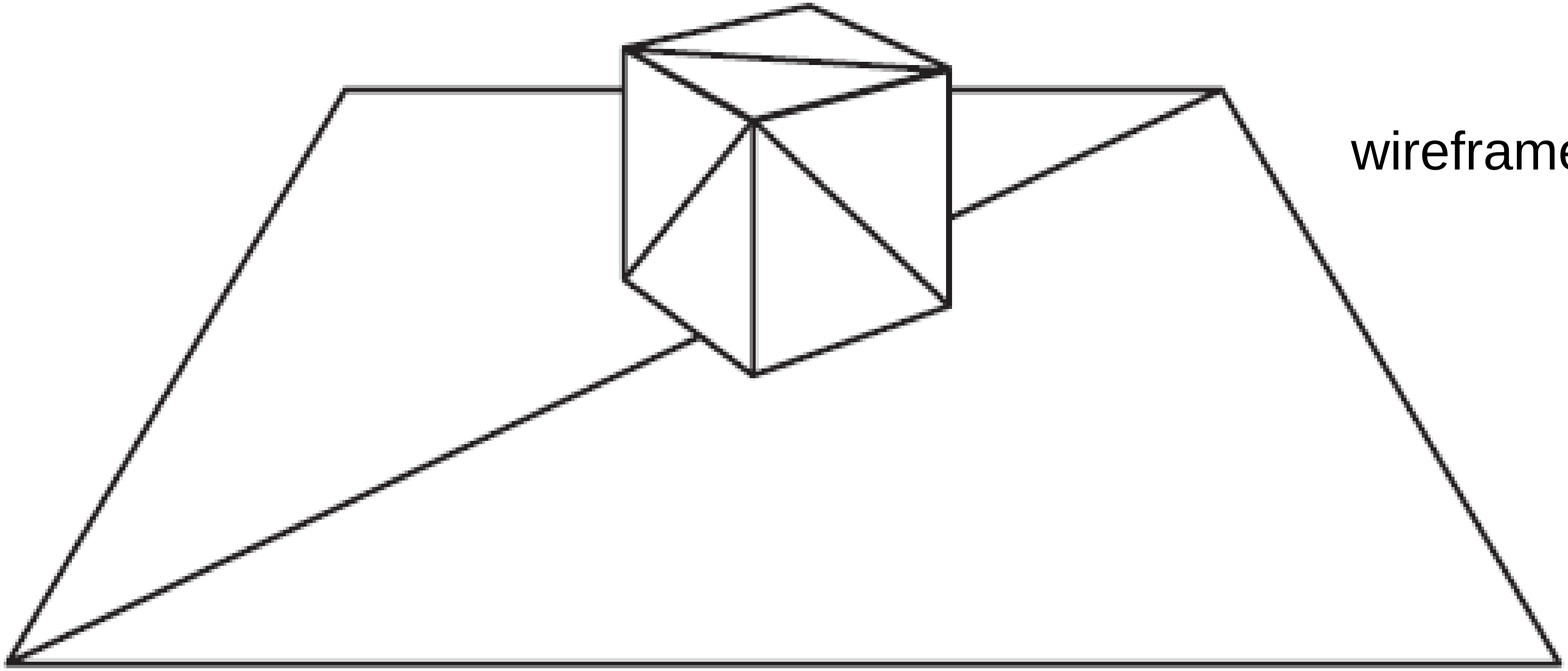
Camera Obscura, Gemma Frisius, 1544

How do we get 2D images of a ~~real~~ synthetic 3D world?

transformation + shading + texture + blending



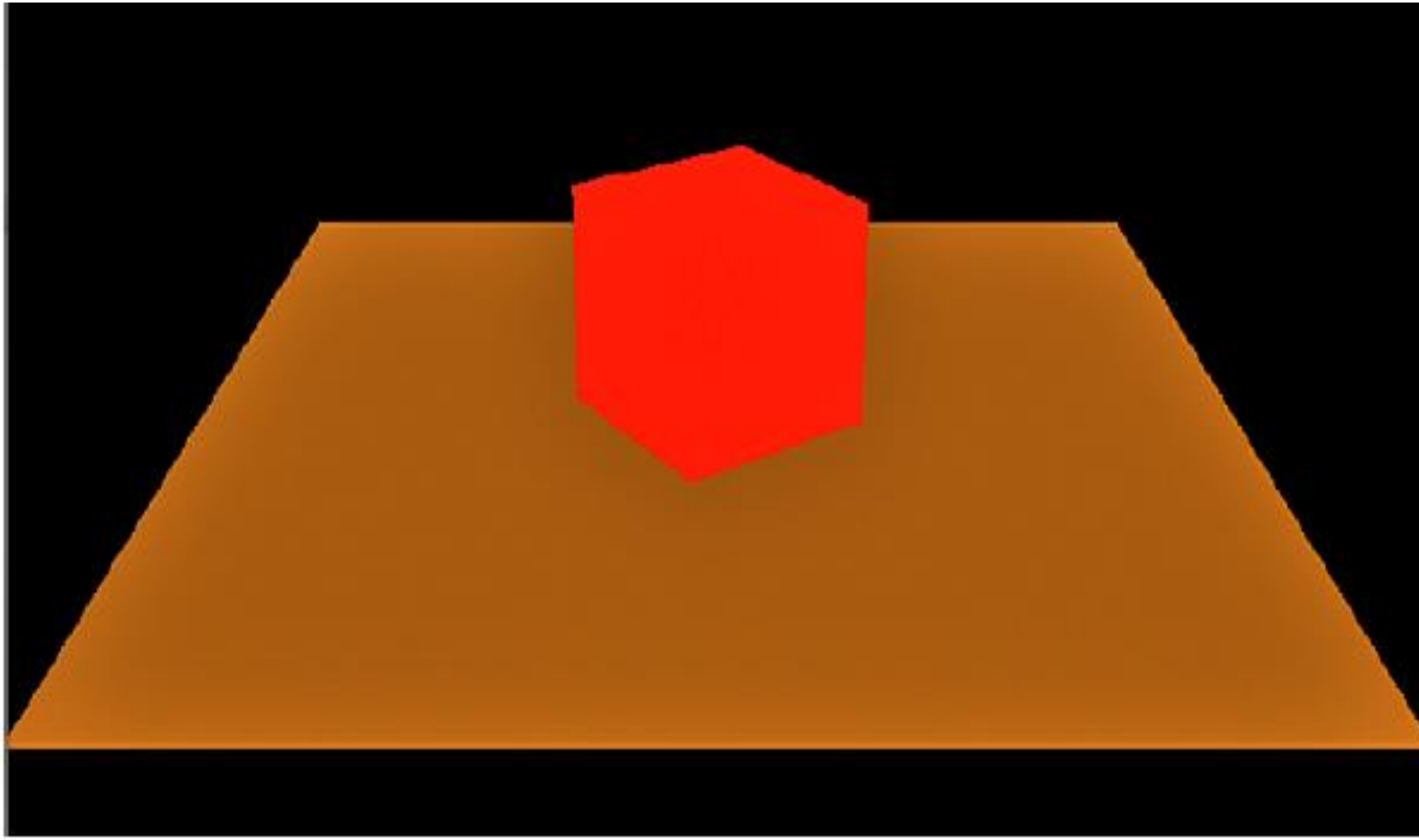
Projected 3D World



wireframe

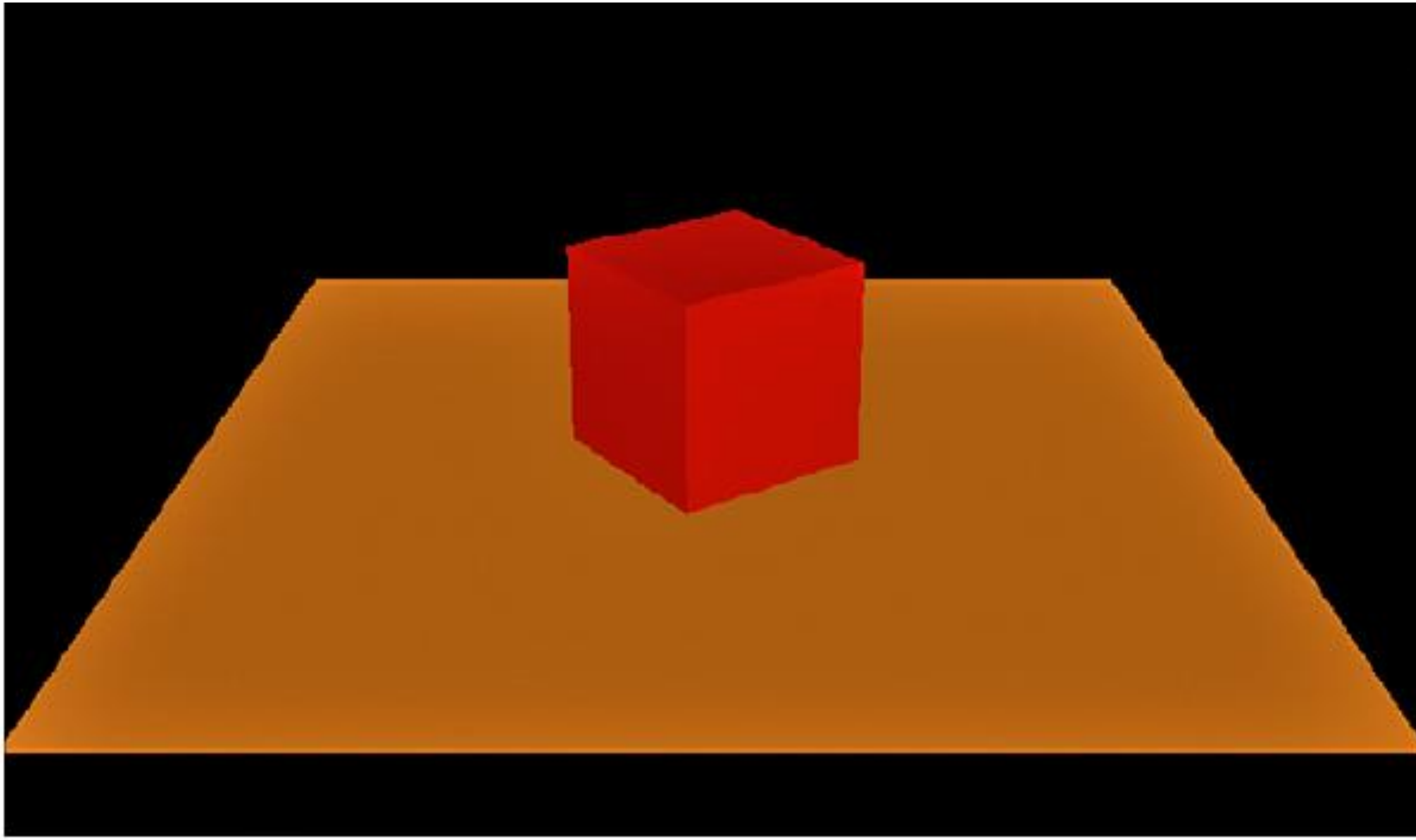
Rasterization

filling with colors



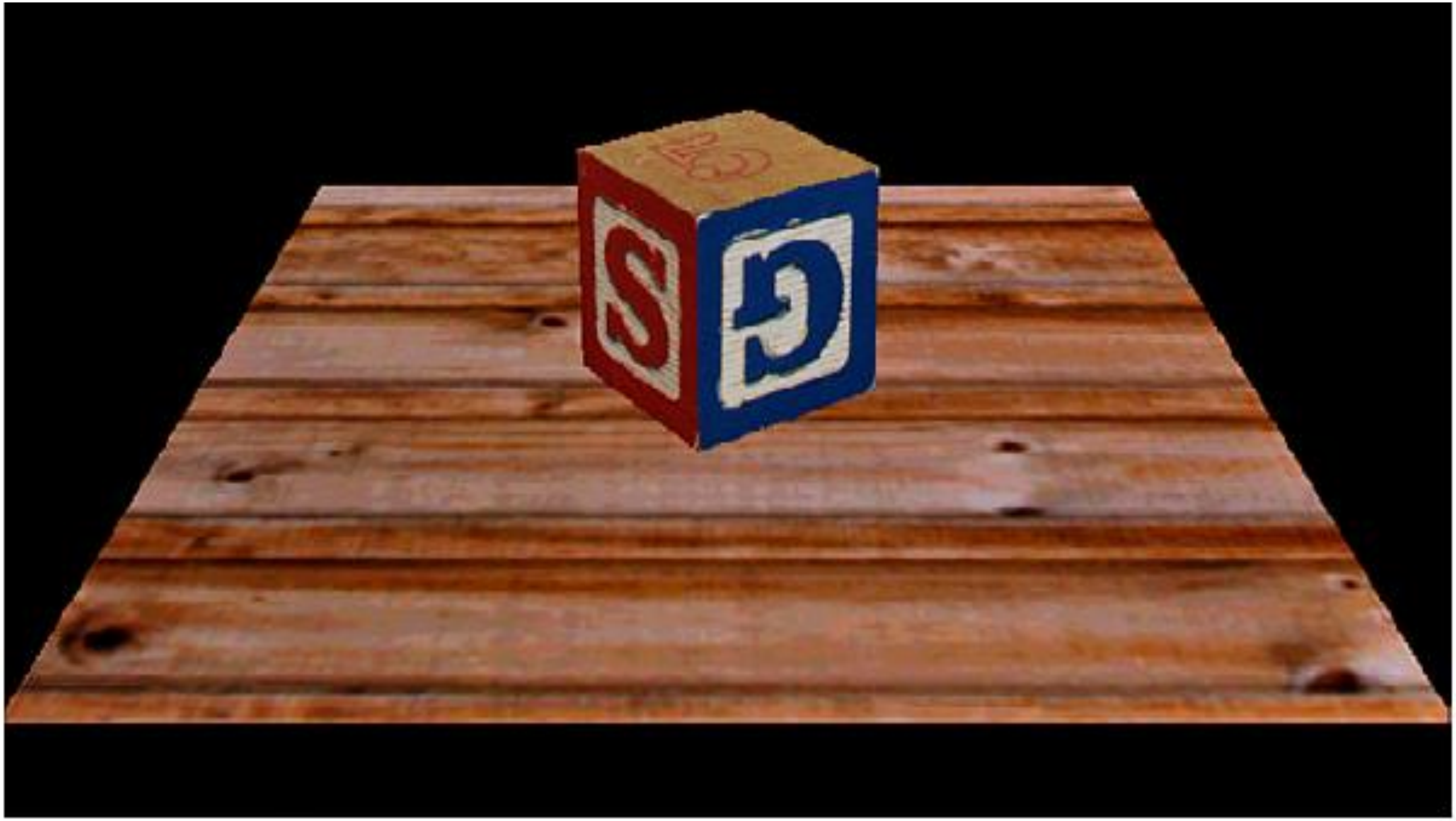
Shading

Varing the color values across the surface (between vertices). Create the **effect** of **light shining** on a red cube



Texture Mapping

A picture that we map to the surface of a triangle or polygon. A texture can simulate an effect that could take thousands of triangles



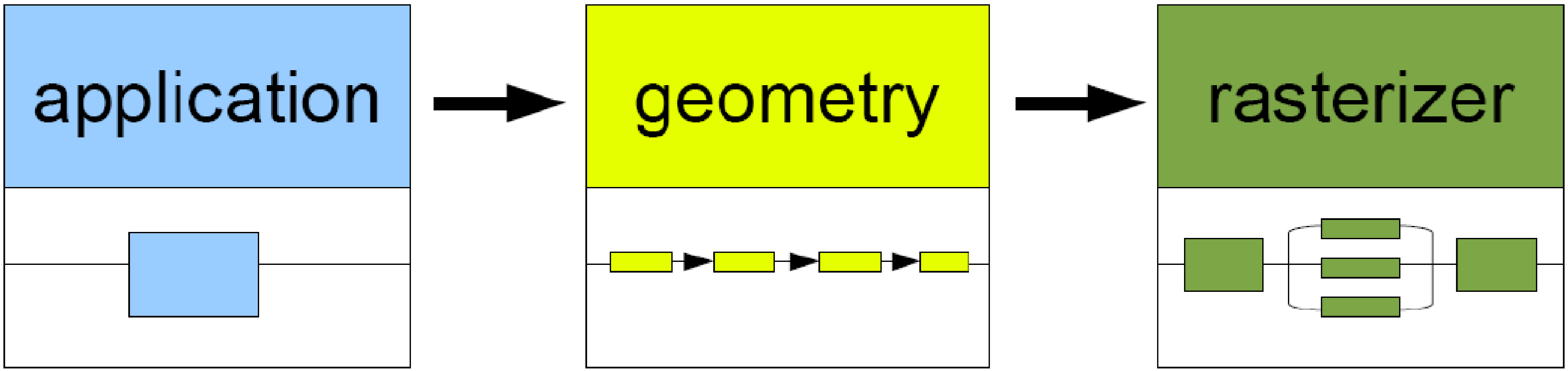
Blending

Allows mixing different colors together.
e.g. create reflections

Basic steps for creating a 2D image out of a 3D world

- Create the 3D world
  - Vertexes and triangles in a 3D space
- Project it to a 2D ‘camera’
  - Use perspective to transform coordinates into a 2D space
- Paint each pixel of the 2D image
  - Rasterization, shading, texturing
  - Will break this into smaller things later on
- Enjoy the super cool image you have created

Pipeline



. **collision** detection
. **animation** global acceleration
. **physics** simulation

process on CPU or GPU

. **transformation**
. **projection**

Computes:
. what is to be drawn
. how should be drawn
. where should be drawn

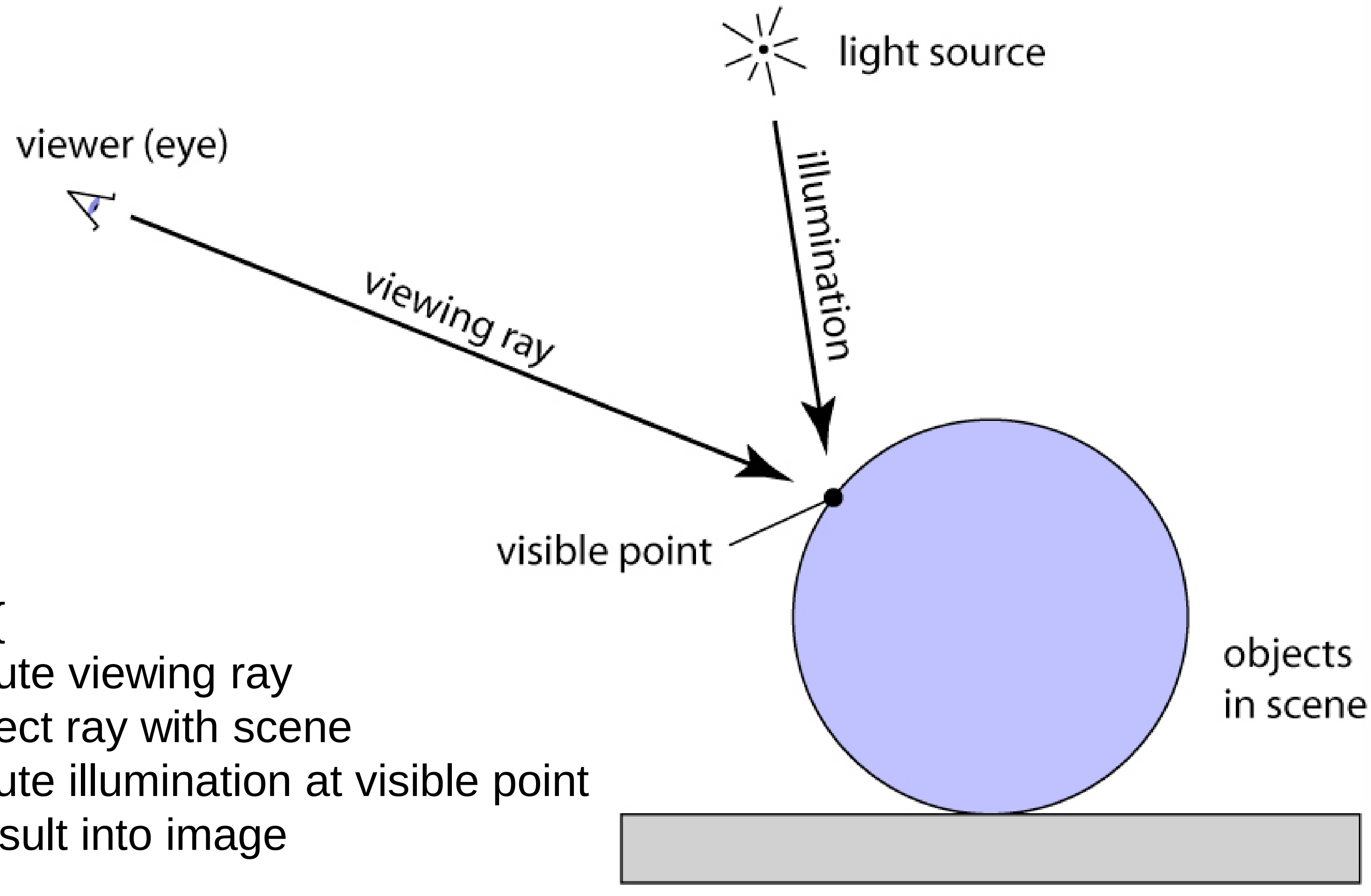
process on GPU

. **draws** images generated by **geometry stage**

process on GPU

Geometry

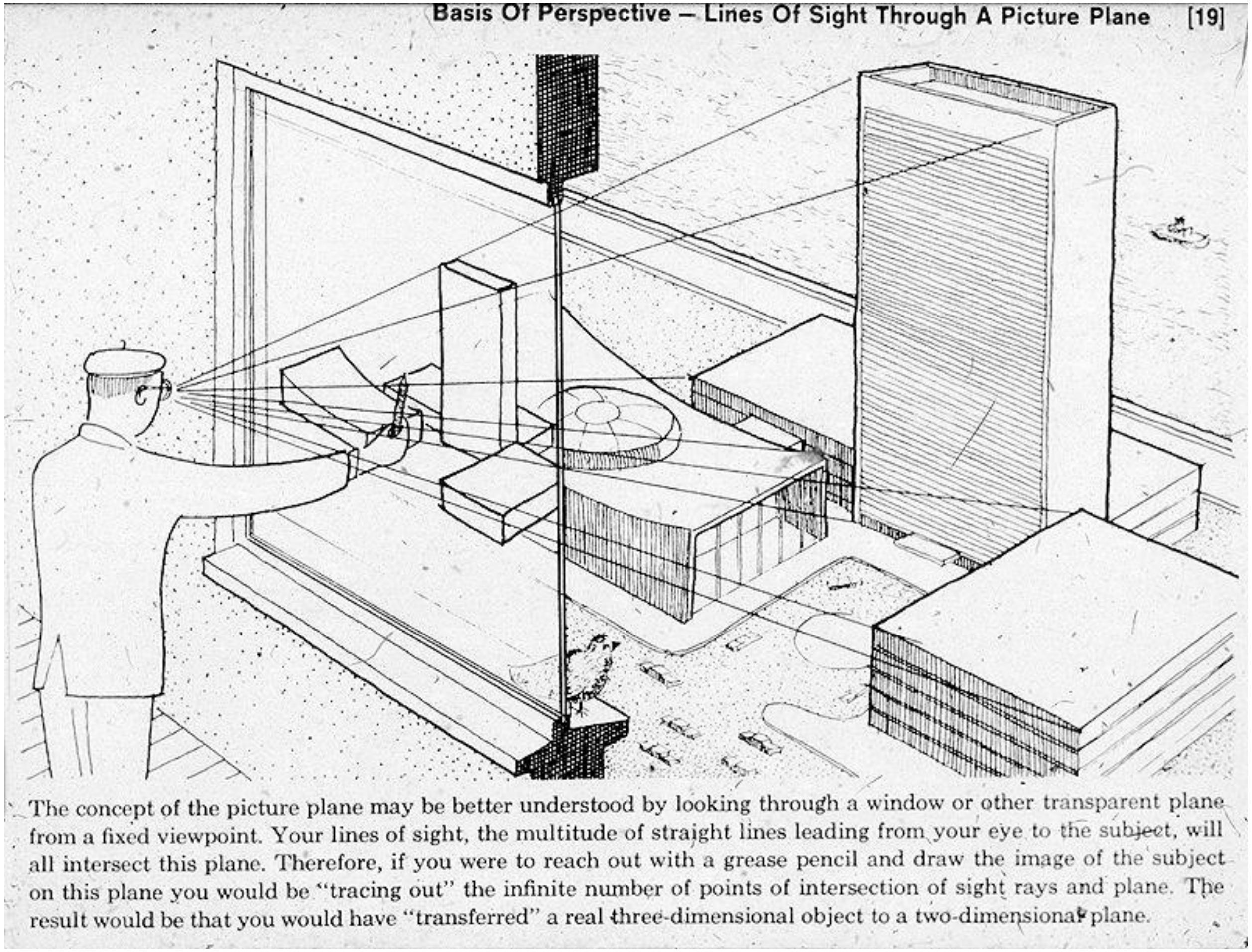
One possibility: Ray tracing



for each pixel {
compute viewing ray
intersect ray with scene
compute illumination at visible point
put result into image
}

Adapted from Steve Marschner, Cornell University

Another one: Projection



Adapted from Steve Marschner, Cornell University

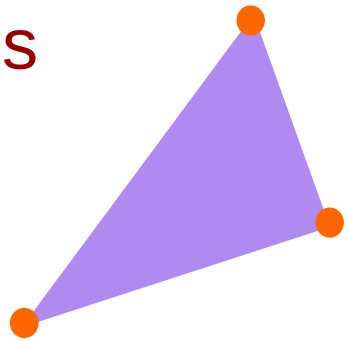
Ray tracing vs. Projection

- Viewing in ray tracing
  – start with image point
  – compute ray that projects to that point
  – do this using geometry
- Viewing by projection
  – start with 3D point
  – compute image point that it projects to
  – do this using transforms
- Inverse processes
  – ray gen. computes the preimage of projection

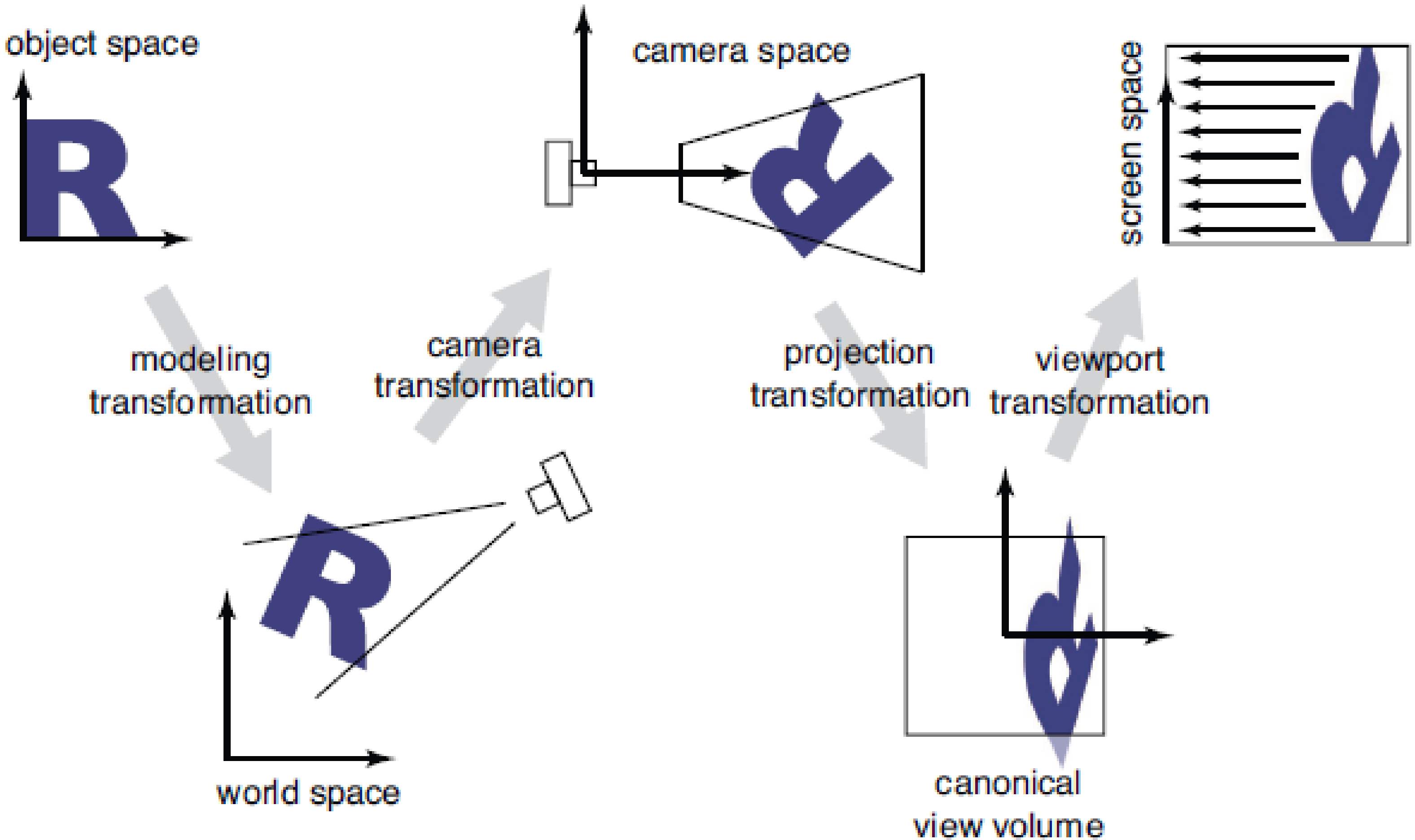
Adapted from Steve Marschner, Cornell University

Representing Geometric Objects

- Geometric objects are represented using vertices
- A vertex is a collection of generic attributes
  - positional coordinates
  - colors
  - texture coordinates
  - any other data associated with that point in space
- Position stored in 4 dimensional homogeneous coordinates
- Vertex data must be stored in *vertex buffer objects* (VBOs)
- VBOs must be stored in *vertex array objects* (VAOs)

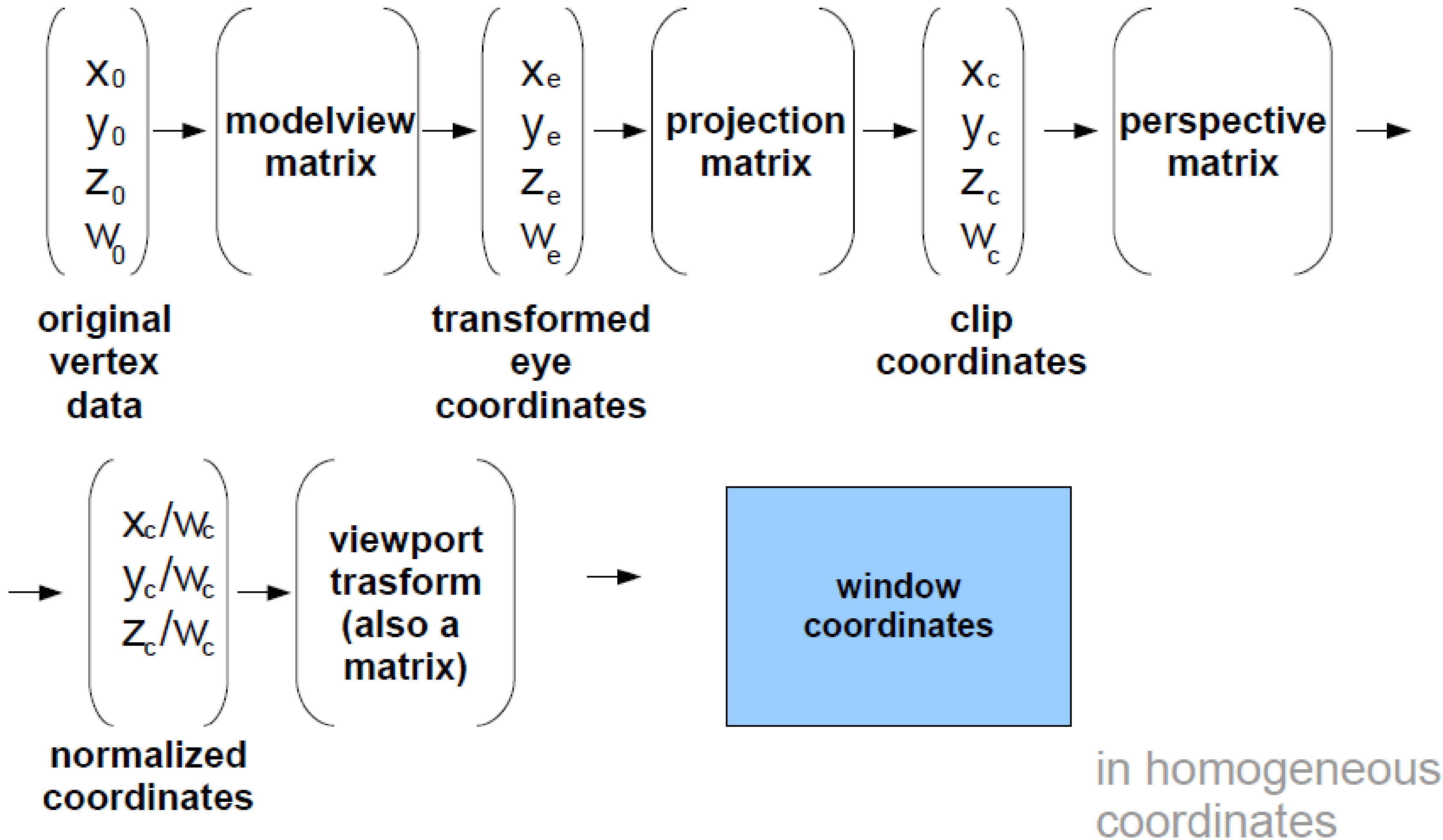


Pipeline of transformations



Adapted from Steve Marschner, Cornell University

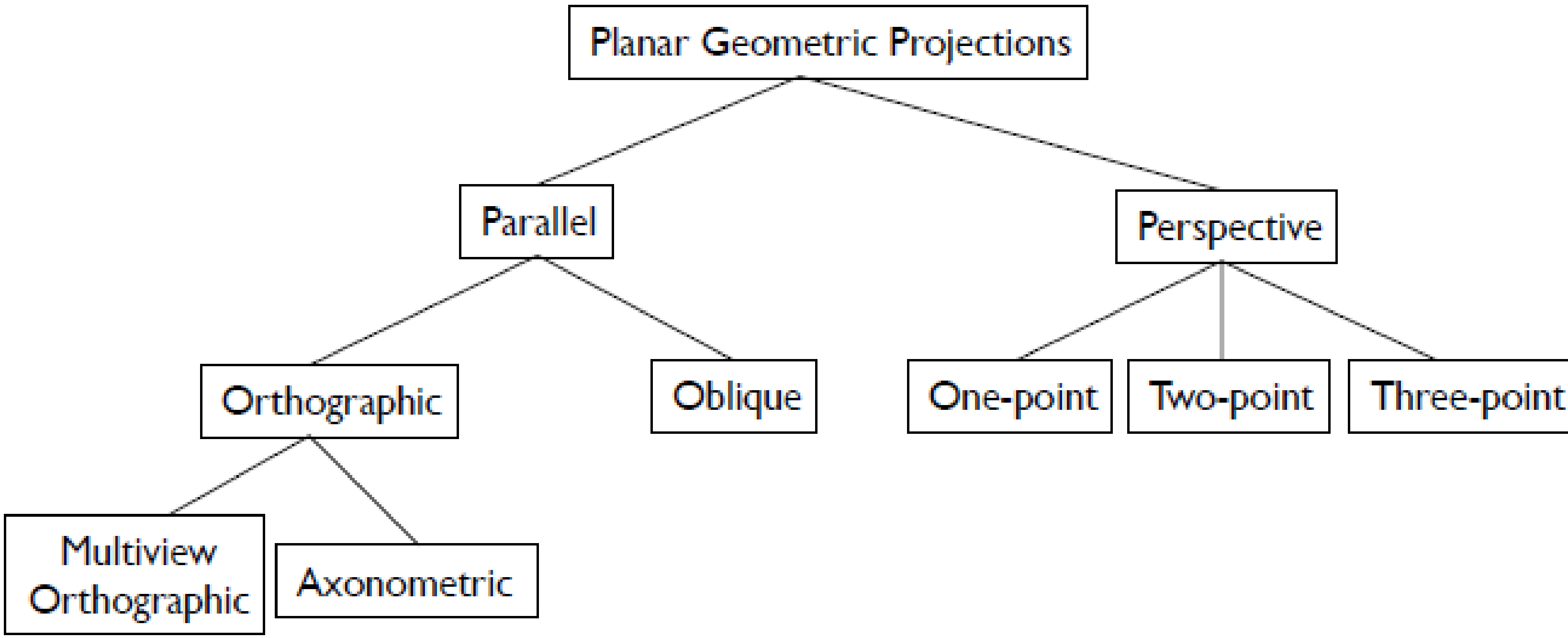
OpenGL transformations pipeline



Projections

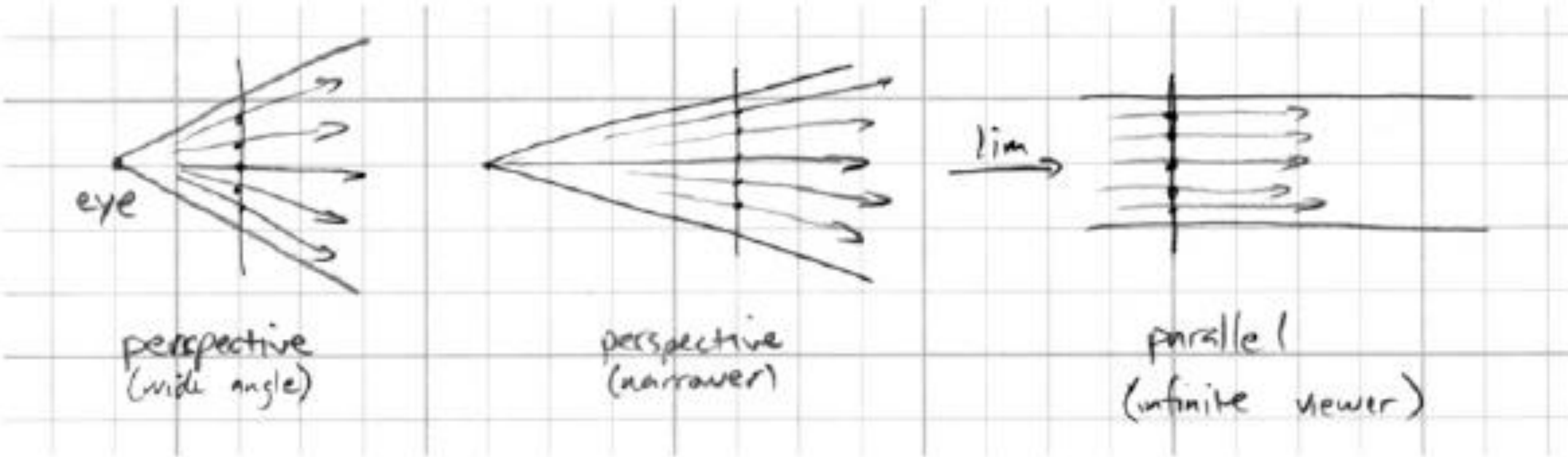
U. PORTO

Classical projections



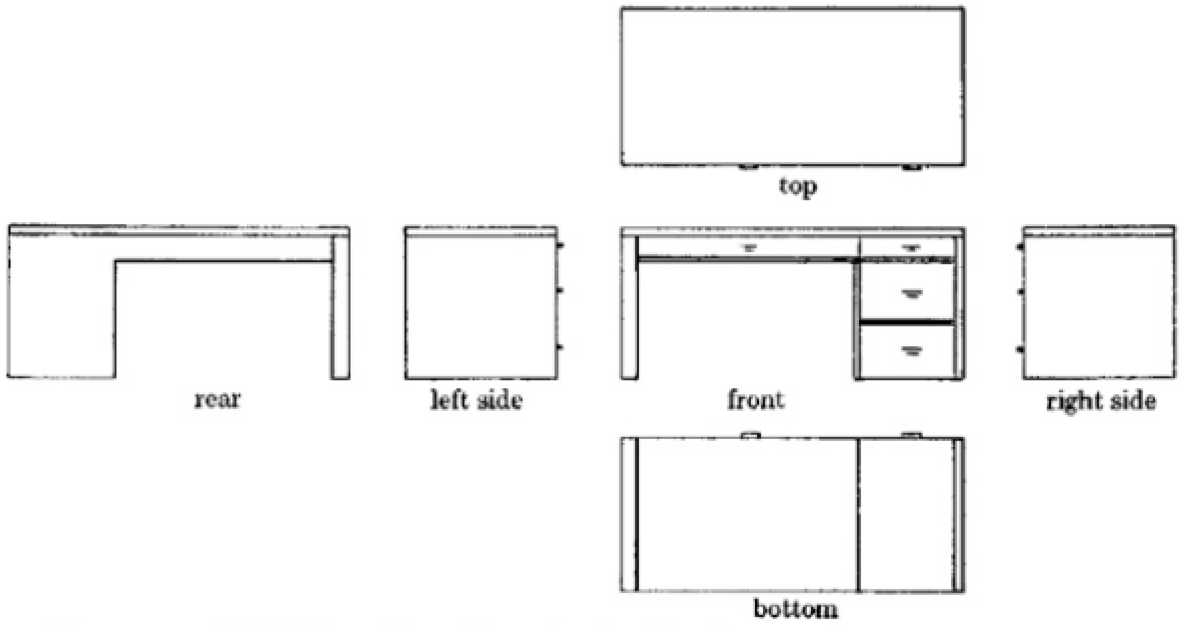
Parallel Projection

- Viewing rays are parallel rather than diverging
  - like a perspective camera that's far away



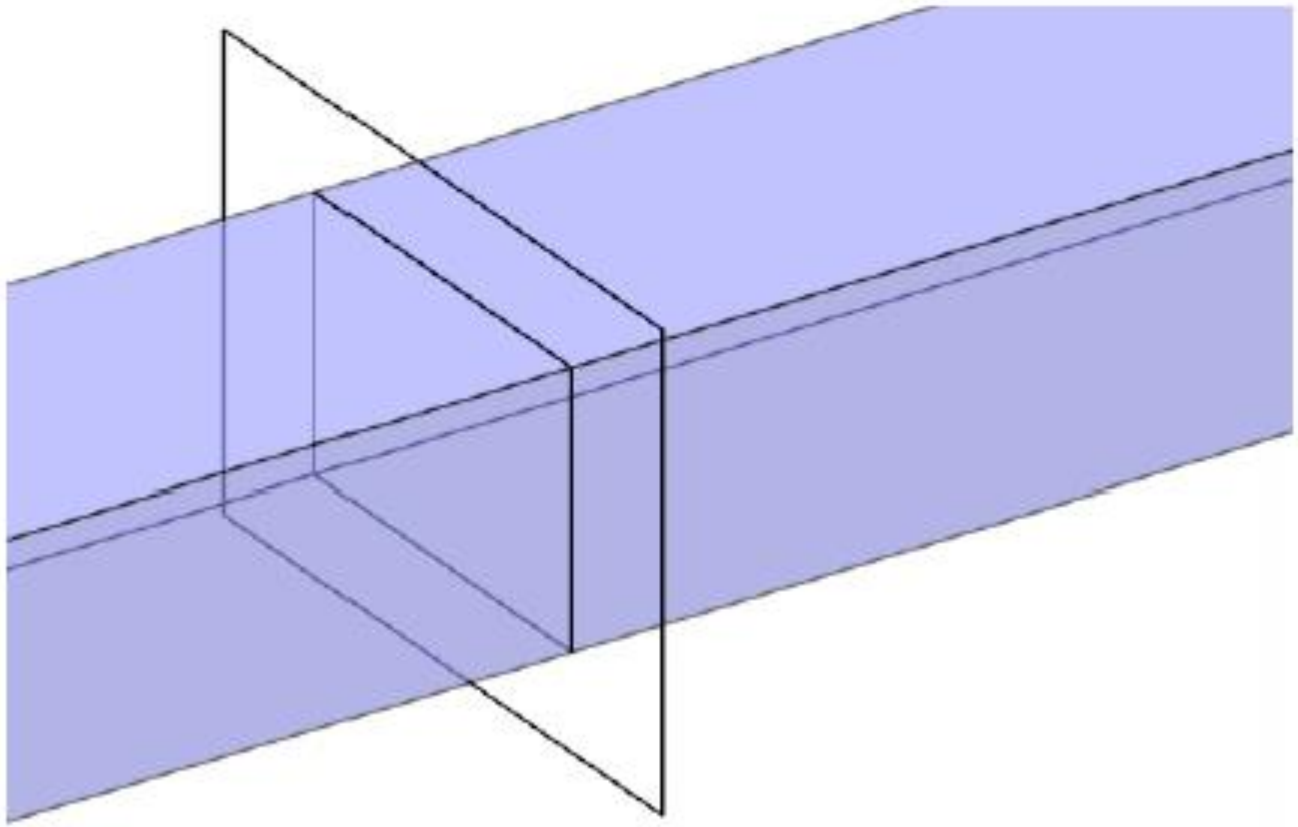
Adapted from Steve Marschner, Cornell University

Multiview orthographic



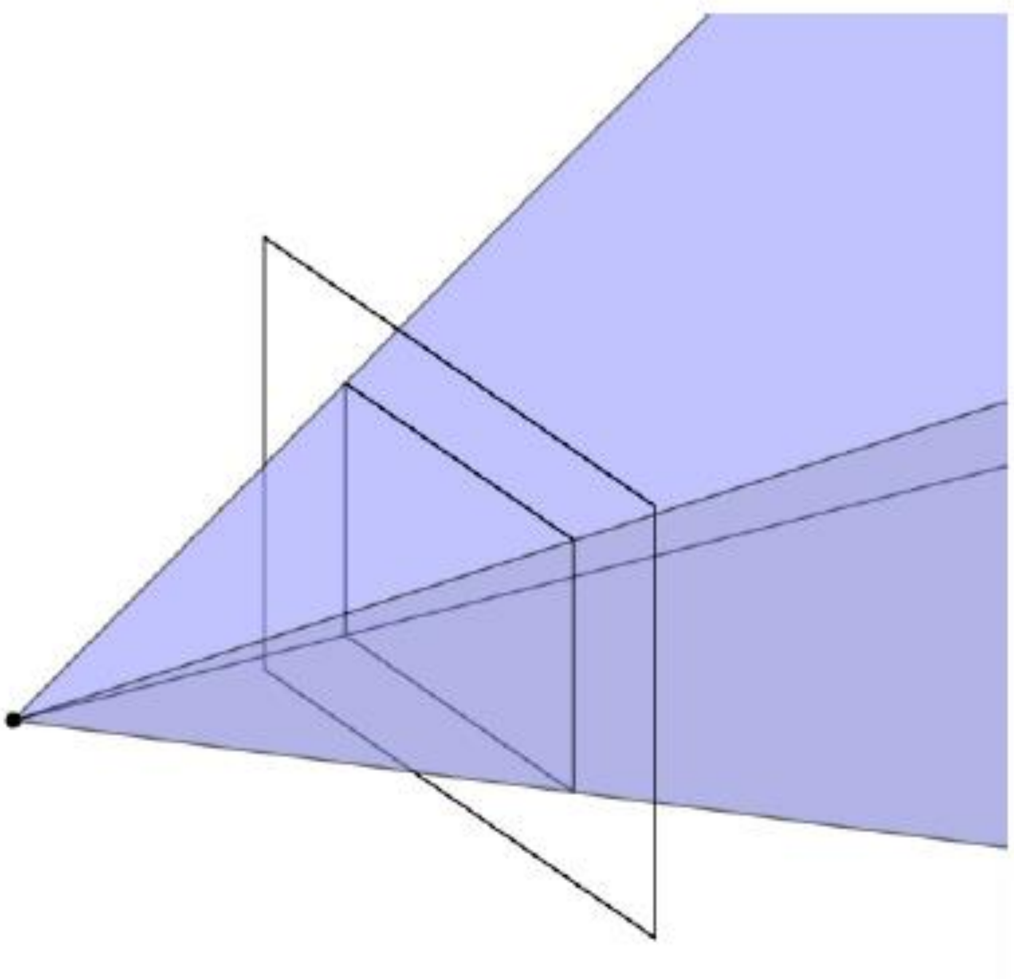
– projection plane parallel to a coordinate plane
– projection direction perpendicular to projection plane

View volume: Orthographic



Adapted from Steve Marschner, Cornell University

View volume: Perspective



Adapted from Steve Marschner, Cornell University

Field of view

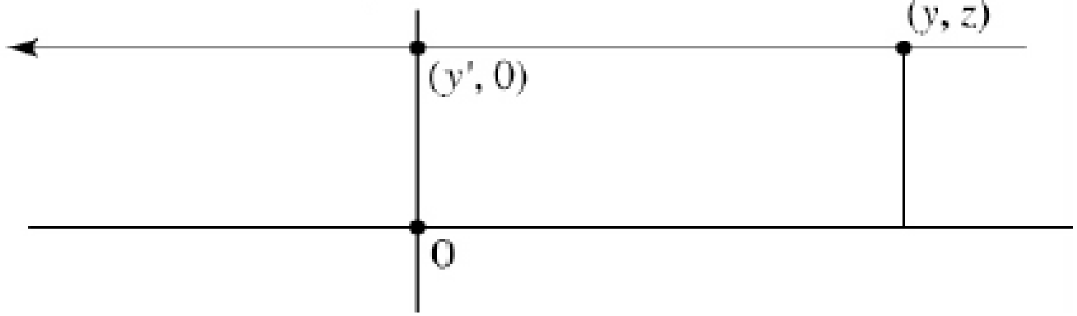
– Angle between the rays corresponding to opposite edges of a perspective image
– Determines 'strength' of perspective effects

camera tilted up: converging vertical lines

Adapted from Steve Marschner, Cornell University

lens shifted up: parallel vertical lines

Orthographic projection

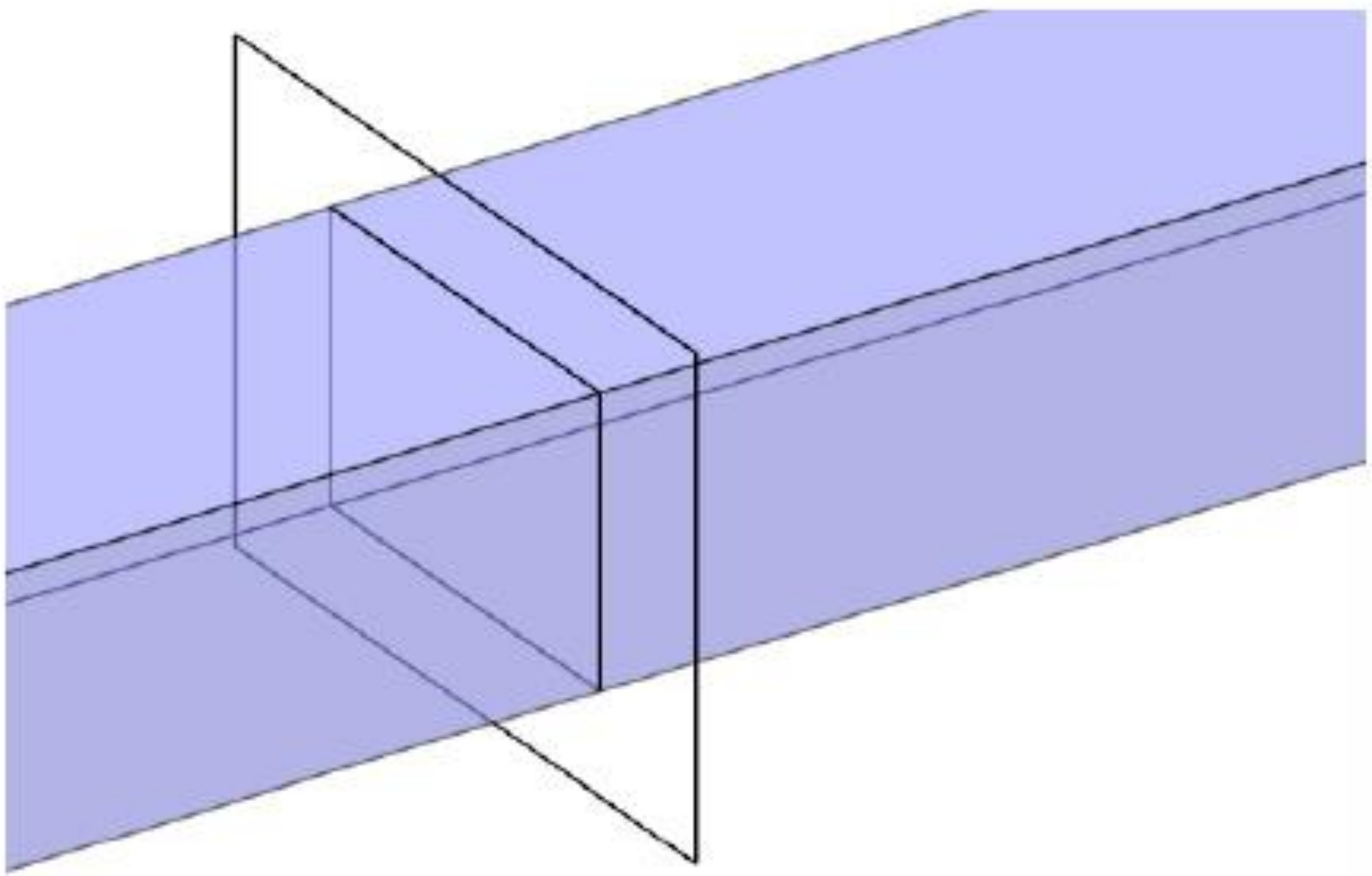


to implement orthographic, just toss out z:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Adapted from Steve Marschner, Cornell University

What about the view volume?



Adapted from Steve Marschner, Cornell University

What about the z direction?

- Two *clipping planes* further constrain the view volume
  - Near plane: parallel to view plane; things between it and the viewpoint will not be rendered
  - Far plane: also parallel; things behind it will not be rendered

Orthographic transformation chain

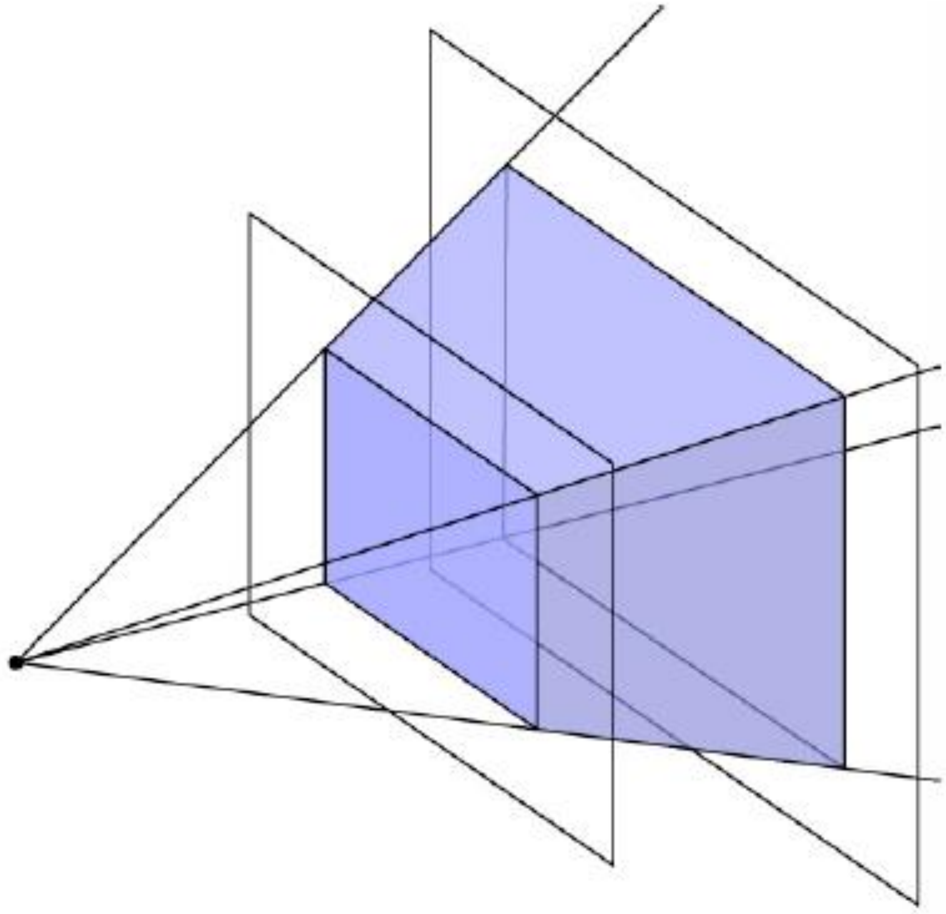
- Start with coordinates in object's local coordinates
- Transform into world coords (modeling transform, M_m)
- Transform into eye coords (camera xf., $M_{cam} = F_c^{-1}$)
- Orthographic projection, M_{orth}
- Viewport transform, M_{vp}

$$\mathbf{p}_s = \mathbf{M}_{\mathrm{vp}}\mathbf{M}_{\mathrm{orth}}\mathbf{M}_{\mathrm{cam}}\mathbf{M}_{\mathrm{m}}\mathbf{p}_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{M}_{\mathrm{m}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

Adapted from Steve Marschner, Cornell University

View volume: Perspective (clipped)



Adapted from Steve Marschner, Cornell University

Perspective transformation chain

- Transform into world coords (modeling transform, M_m)
- Transform into eye coords (camera xf., $M_{\text{cam}} = F_c^{-1}$)
- Perspective matrix, P
- Orthographic projection, M_{orth}
- Viewport transform, M_{vp}

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{orth}}\mathbf{P}\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}}\mathbf{p}_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

Summary: Projection

- Different types of projection
  - Orthographic
  - Perspective
- Integrate nicely into the transformation chain
- Other elements:
  - Viewing transform
  - Viewport transform

Rasterization

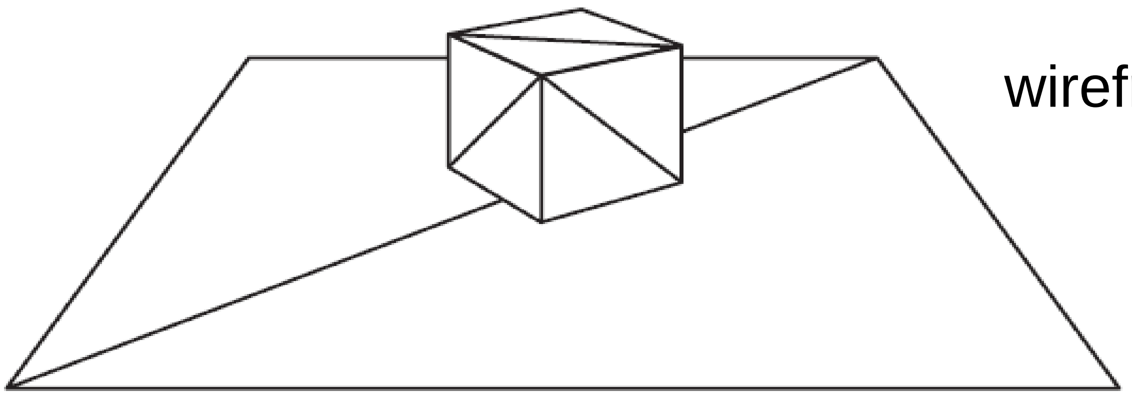
U. PORTO

Basic steps for creating a 2D image out of a 3D world

- Create the 3D world
  - Vertexes and triangles in a 3D space
- Project it to a 2D ‘camera’
  - Use perspective to transform coordinates into a 2D space
- Paint each pixel of the 2D image
  - **Rasterization**, shading, texturing
  - Will break this into smaller things later on
- Enjoy the super cool image you have created

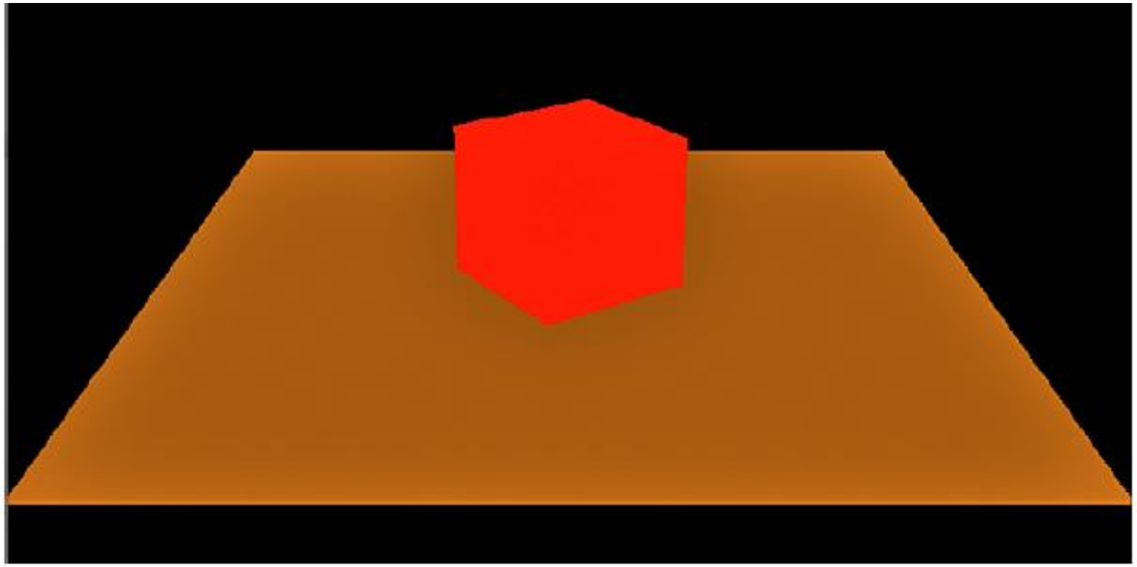
Rasterization

rasterization:

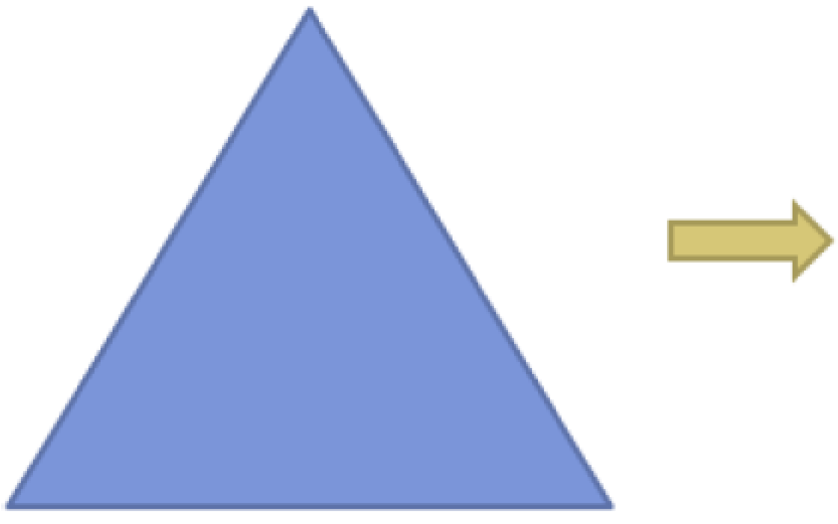


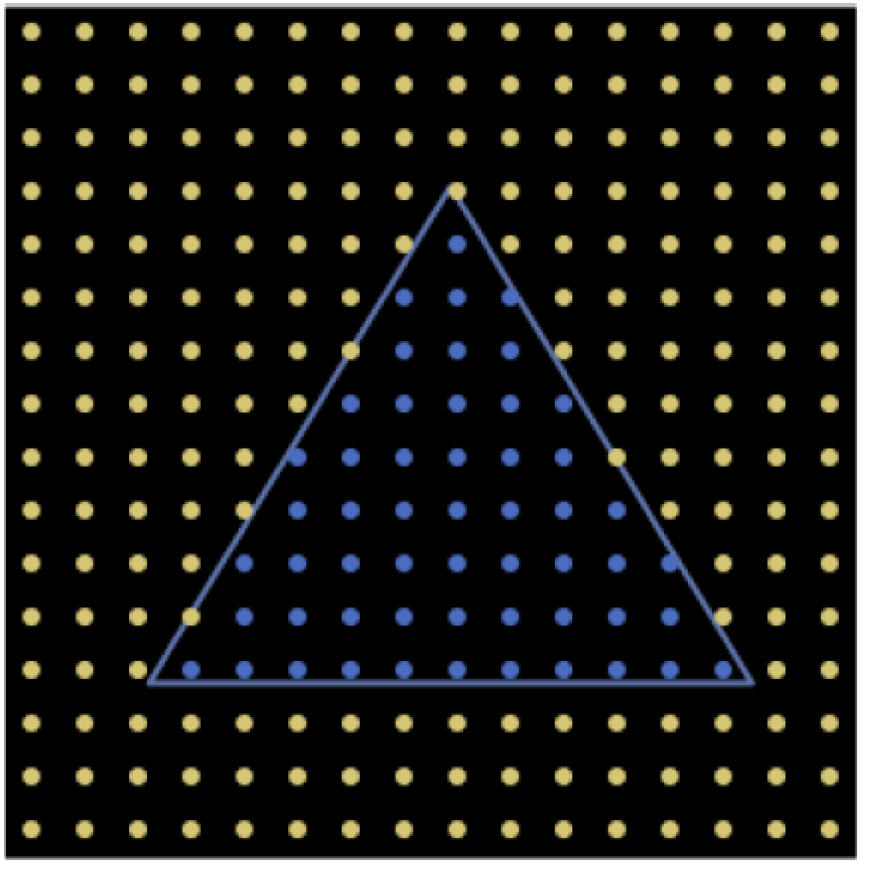
wireframe

filling with colors



Rasterization





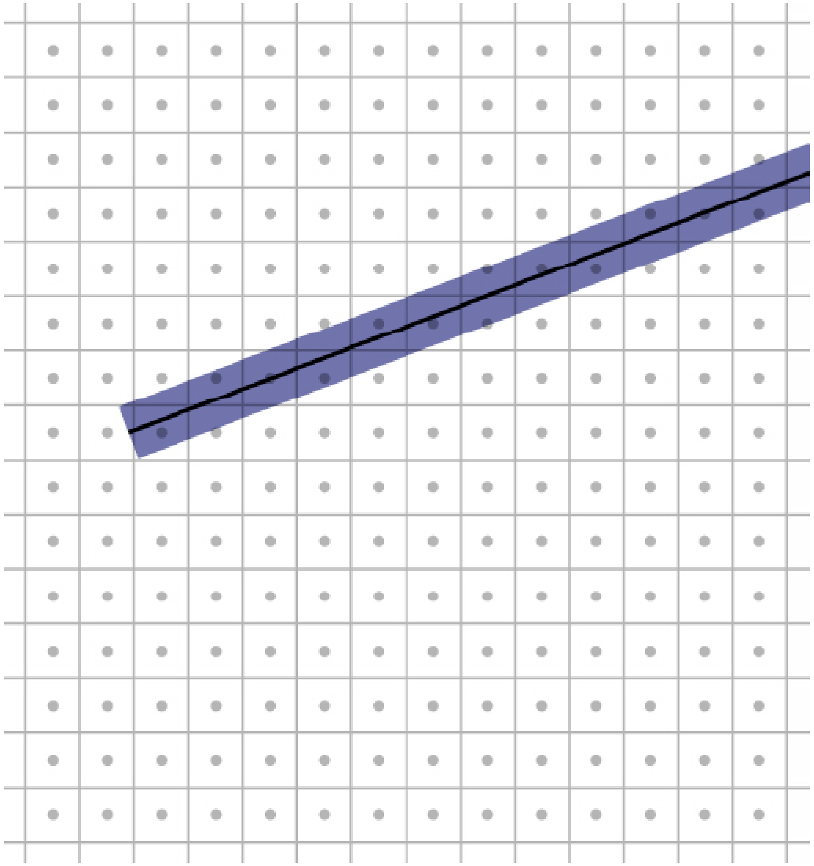
Slide by Ron Fedkiw, Stanford University

Primitives

- Only three!
  - Points
  - Line segments
  - Triangles
- How do I rasterize them?
  - Points are simple
  - Lines?
  - Triangles?

Rasterizing lines

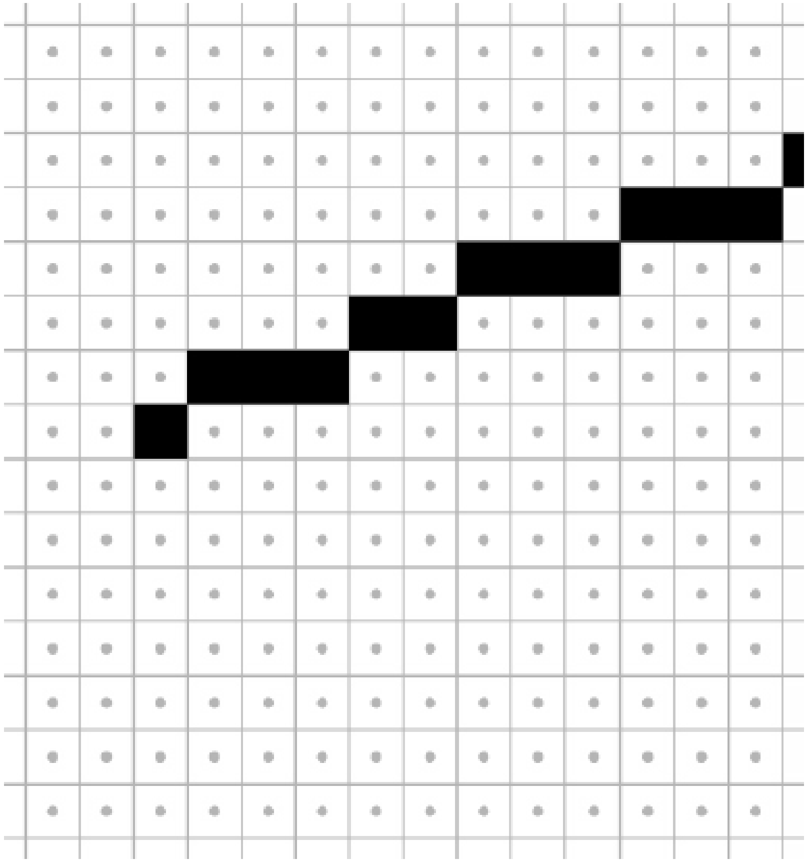
- Lines are defined by two points
  - Projected into my 2D screen from my 3D world
- Consider it a rectangle
  - So that it occupies a non-zero area



Adapted from Steve Marschner, Cornell University

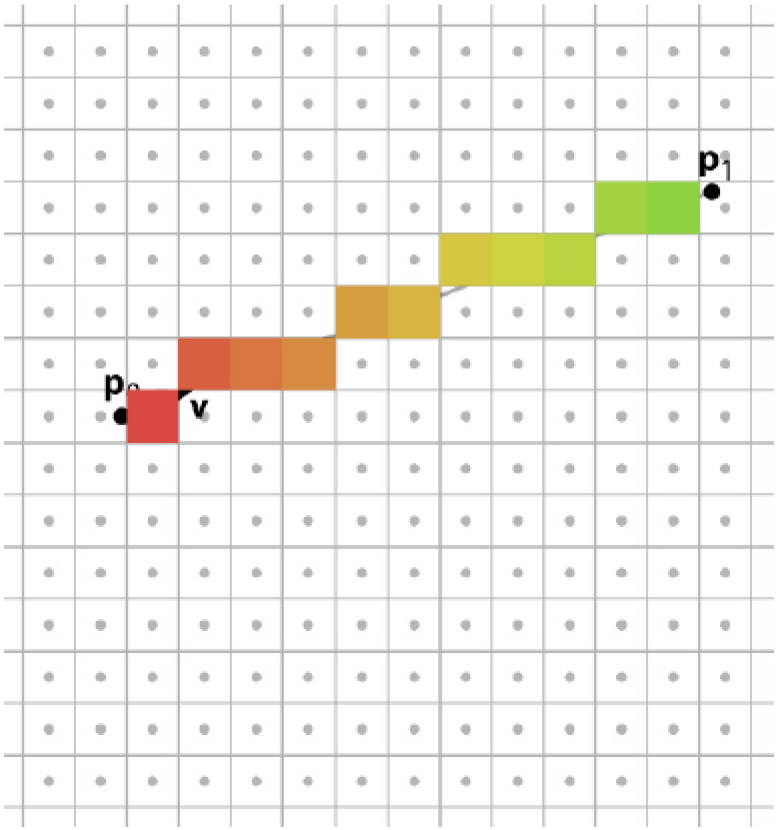
Bresenham lines (midpoint alg.)

- Idea:
  - Define line width parallel to pixel grid
- What does this mean?
  - Turn on the single nearest pixel in each column



Interpolation along lines

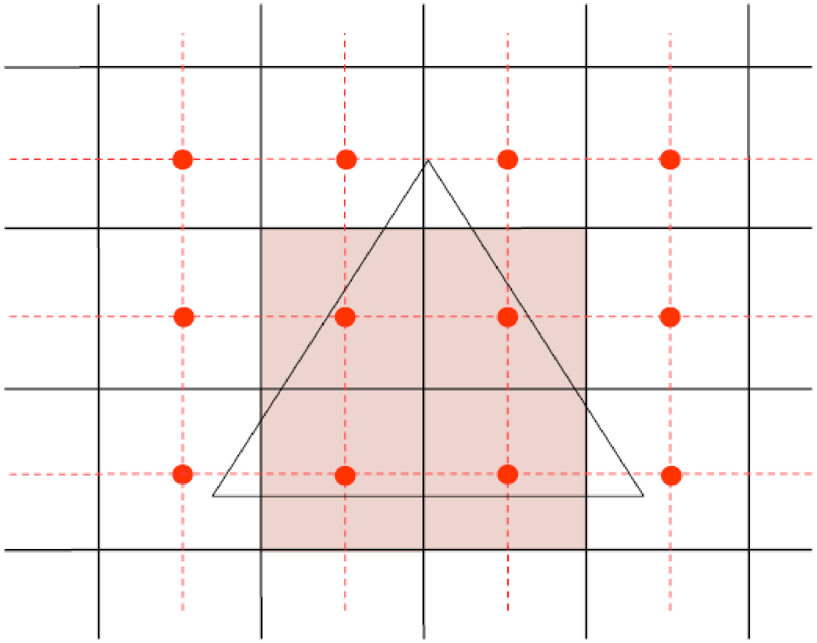
- We don't want to simply know which pixels are on the line
  - Boolean
- Vertexes hold attributes
  - Ex: Color
- We want these to vary smoothly along the line
  - Linear interpolation



Adapted from Steve Marschner, Cornell University

Rasterizing triangles

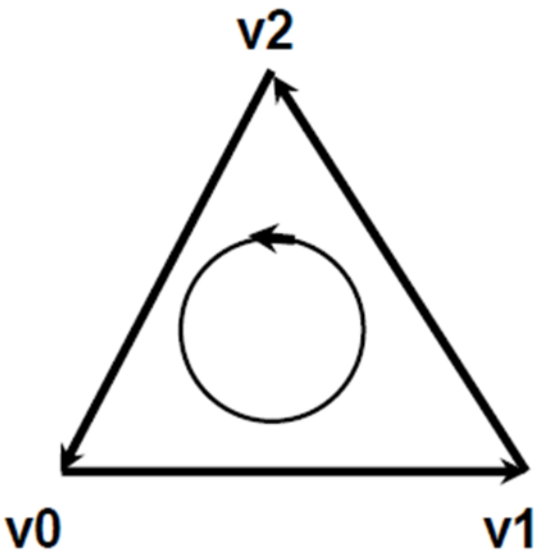
- Pixel belongs to the triangle if its center is inside the triangle
- Need two things:
  - Which pixels belong to the triangle?
  - How do we interpolate values from 3 vertexes?



Adapted from Ron Fedkiw, Stanford University

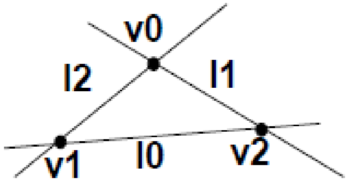
Using directed lines

- Point is inside the triangle if it is <u>on the left of three directed lines</u>
  - They could be on the right too…
- How do we build a simple test for this?



Point inside triangle test

```
makeline( vert& v0, vert& v1, line& l )
{
  l.a = v1.y - v0.y;
  l.b = v0.x - v1.x;
  l.c = -(l.a * v0.x + l.b * v0.y);
}
rasterize( vert v[3] )
{
  line l0, l1, l2;
  makeline(v[0],v[1],l2);
  makeline(v[1],v[2],l0);
  makeline(v[2],v[0],l1);
  for( y=0; y<YRES; y++ ) {
    for( x=0; x<XRES; x++ ) {
      e0 = l0.a * x + l0.b * y + l0.c;
      e1 = l1.a * x + l1.b * y + l1.c;
      e2 = l2.a * x + l2.b * y + l2.c;
      if( e0<=0 && e1<=0 && e2<=0 )
        fragment(x,y);
    }
  }
}
```



Illumination

Basic steps for creating a 2D image out of a 3D world

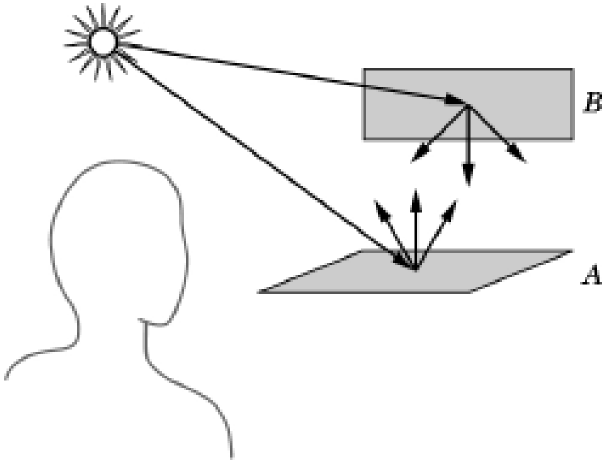
- Create the 3D world
  - Vertexes and triangles in a 3D space
- Project it to a 2D ‘camera’
  - Use perspective to transform coordinates into a 2D space
- Paint each pixel of the 2D image
  - **Rasterization, shading,** texturing
  - Will break this into smaller things later on
- Enjoy the super cool image you have created

Illumination: main concepts

light sources **emit light**

. color spectrum

. position and direction



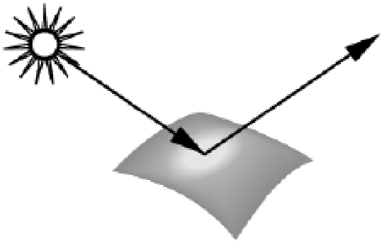
surfaces **reflect light**

. reflectance

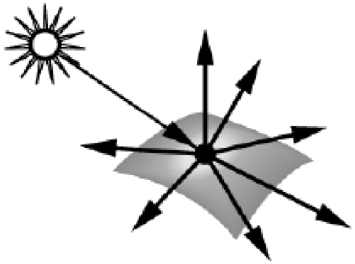
. geometry

. transmission

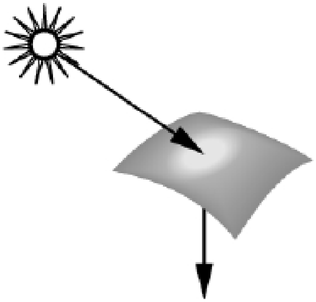
. absorption



specular



diffuse



transparency

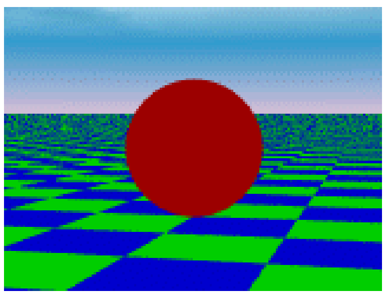
Illumination: main concepts

Illumination determined by the interaction of the

light source + **the surface**

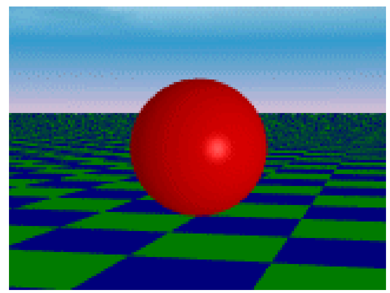
Illumination: types of lights

ambient

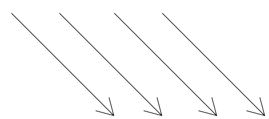


Indirect illumination

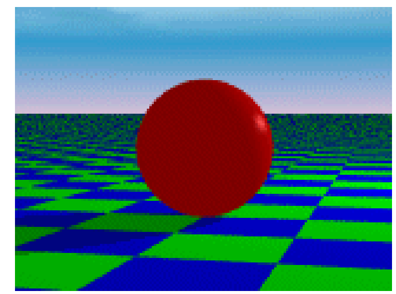
directional



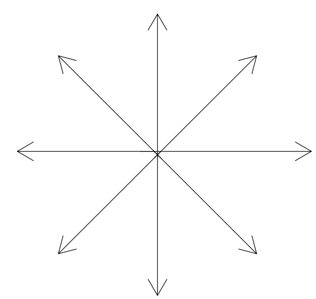
sun



point



light bulb

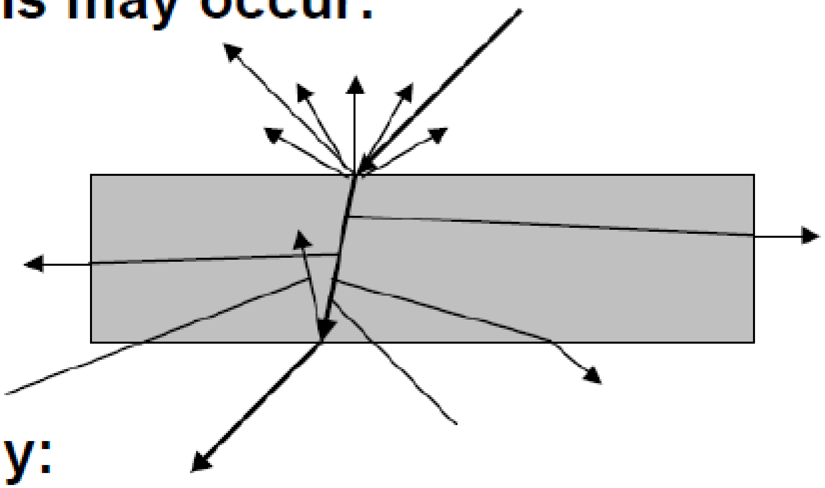


How does light interact with a surface?

Three types of interactions

When light makes contact with a material, three types of interactions may occur:

- **Reflection**
- **Absorption**
- **Transmittance**



From conservation of energy:

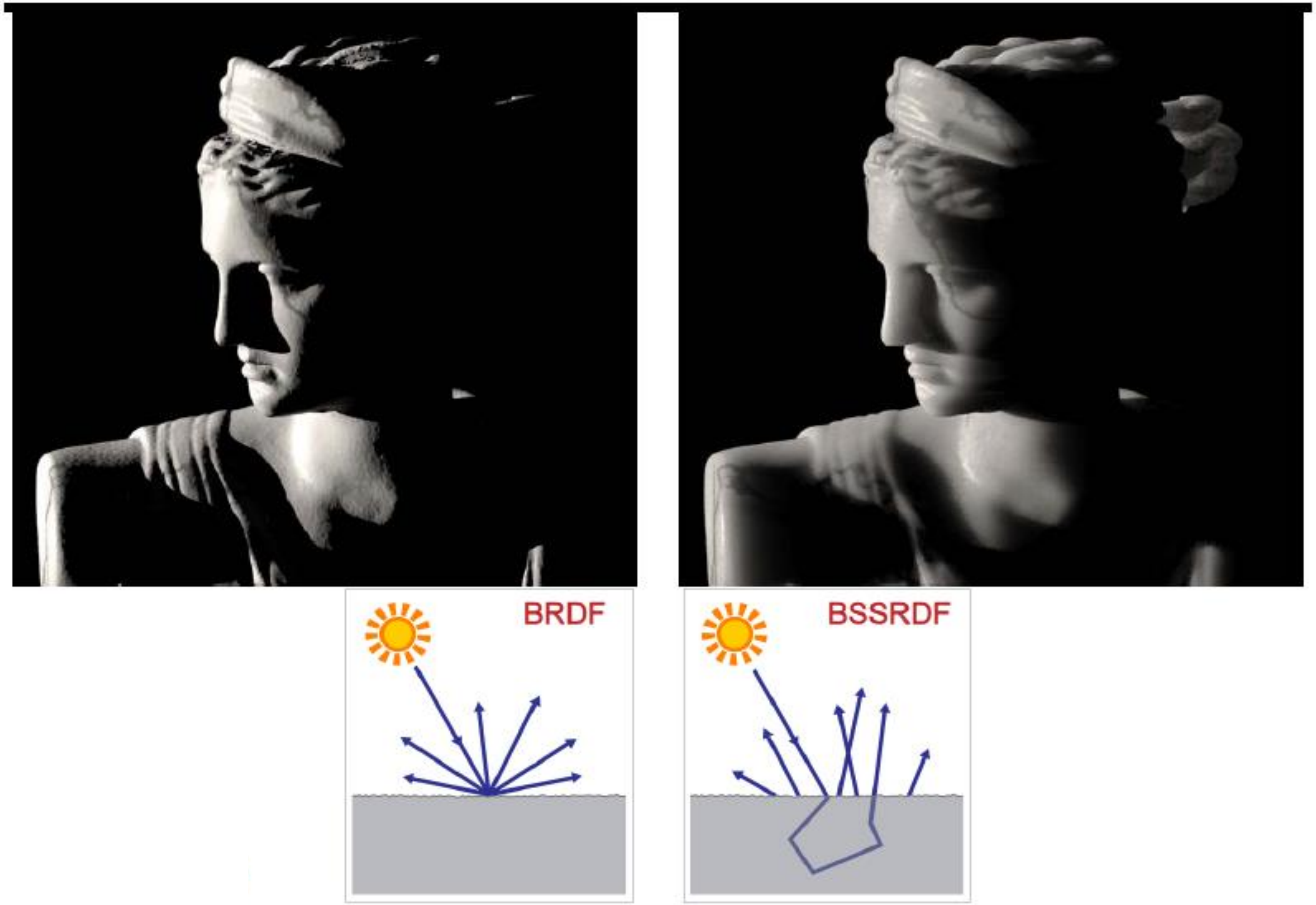
$$\text{light incident at surface} = \text{light reflected} + \text{light absorbed} + \text{light transmitted}$$

Opaque object: the majority of incident light is either reflected or absorbed – transmitted light≈0

Translucent object: significant light transmission

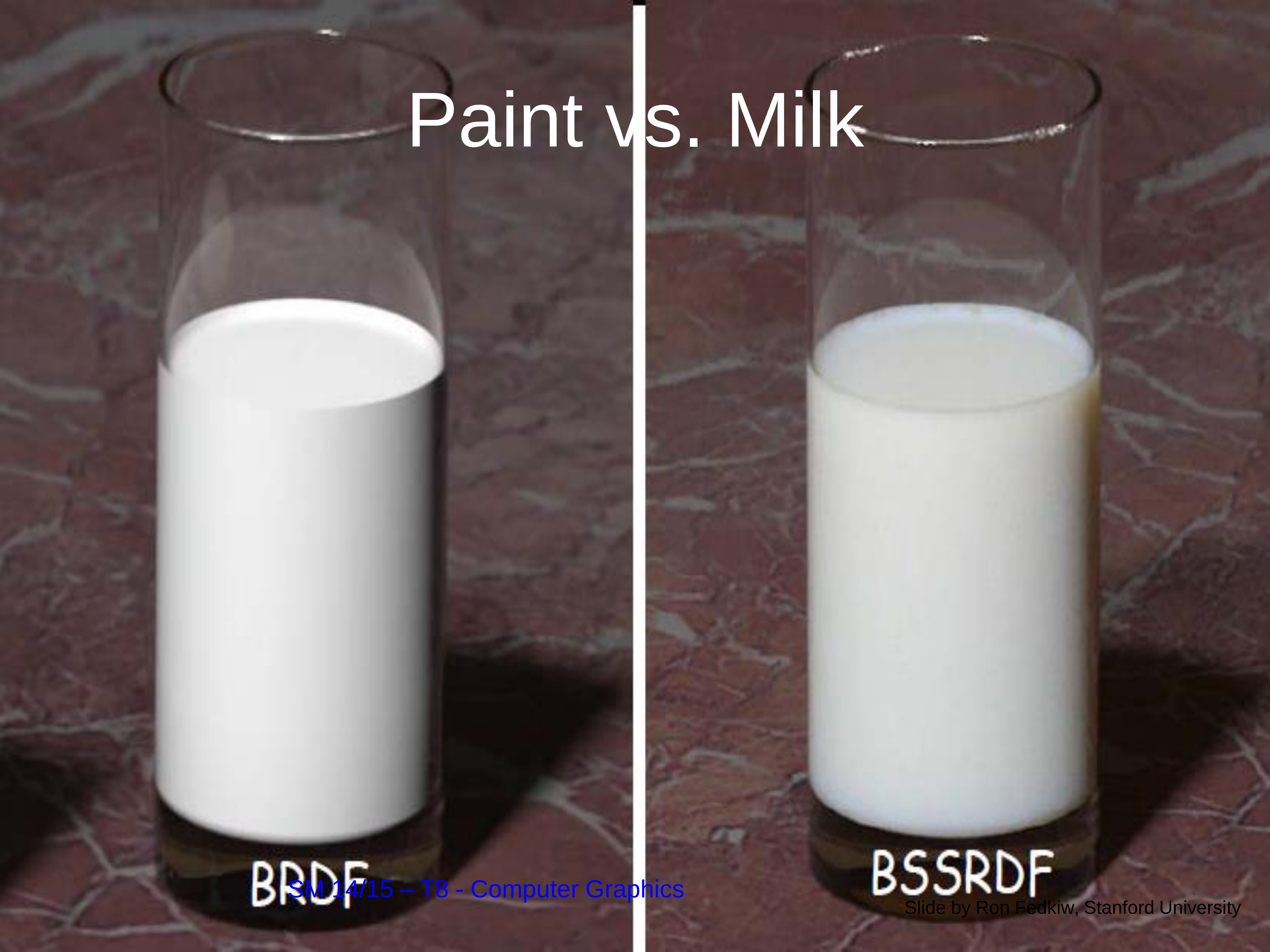
Slide by Ron Fedkiw, Stanford University

Opaque vs Translucent



Slide by Ron Fedkiw, Stanford University

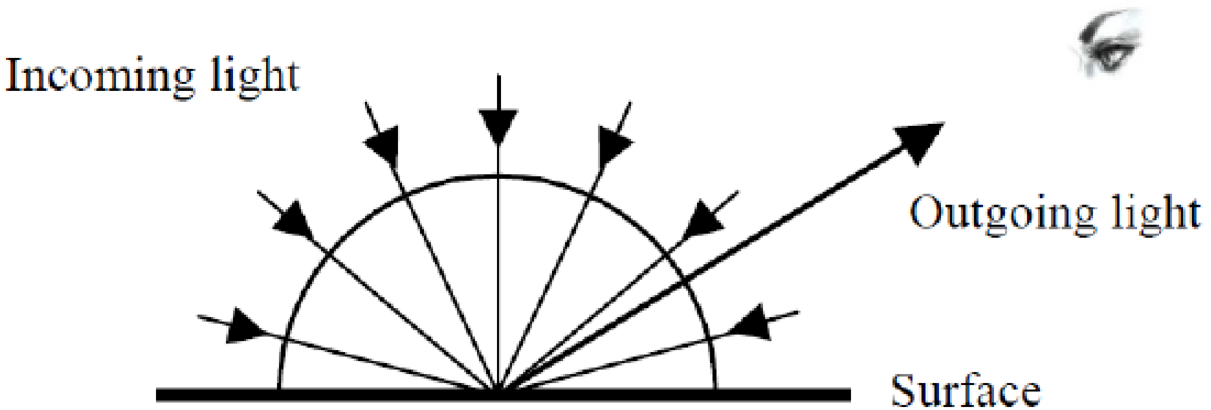
Paint vs. Milk



Slide by Ron Fedkiw, Stanford University

The lighting equation

In the real world, the entire environment surrounding a surface in a scene contributes to the illumination of every surface point



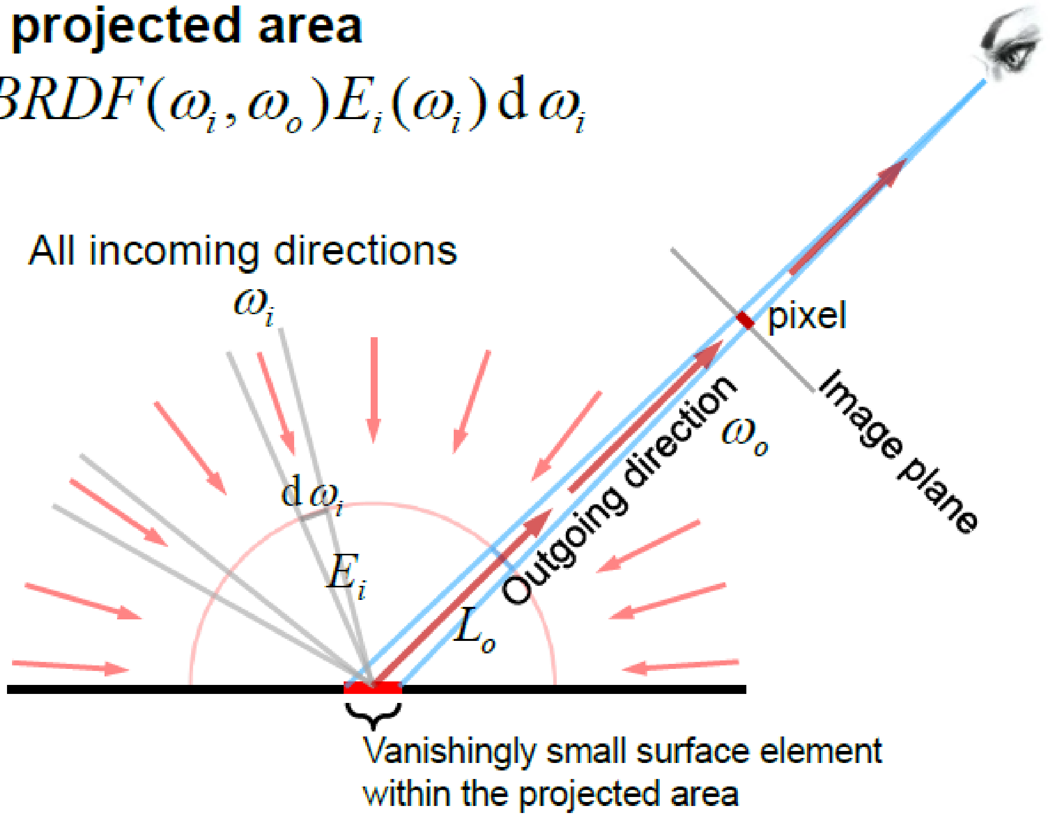
The amount of light reflected in an outgoing direction is the integral of the amount of light reflected in that direction due to light from every incoming direction. Discretely put, we have $L_o = \sum_{i \in \text{in}} L_{o \text{ due to } i}(\omega_i, \omega_o)$

Slide by Ron Fedkiw, Stanford University

How to colour a pixel?

For each pixel in the image plane, need to integrate the BRDF across all incoming directions for every point in the projected area

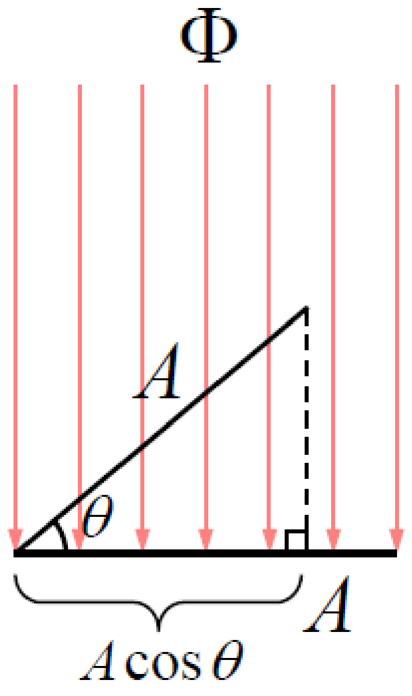
$$L_o = \int_{i \in \text{in}} BRDF(\omega_i, \omega_o) E_i(\omega_i)\, \mathrm{d}\,\omega_i$$



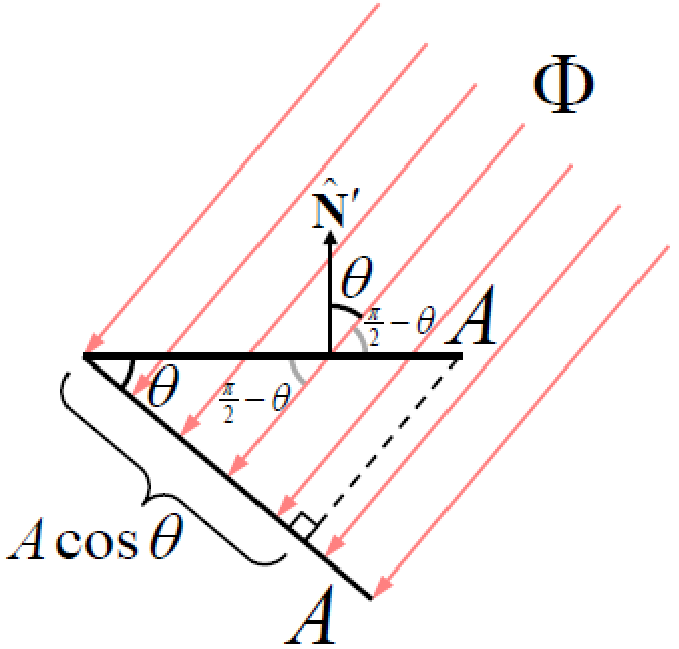
Slide by Ron Fedkiw, Stanford University

Irradiance

Power per unit surface area $E \equiv \frac{d\Phi}{dA}$



$$E_{\text{tilted}} = \frac{\left(\frac{A\cos\theta}{A}\right)\Phi}{A} = \frac{\Phi\cos\theta}{A} = E\cos\theta$$



Note how the irradiance decreases as you tilt the object, since it fits into a smaller solid angle

Slide by Ron Fedkiw, Stanford University

some basics you MUST know

Types of Lights

1. Ambient

2. Diffuse

3. Specular

4. Emissive: color of a surface adds intensity to the object, but is unaffected by any light sources. Does not introduce any additional light into the overall scene.

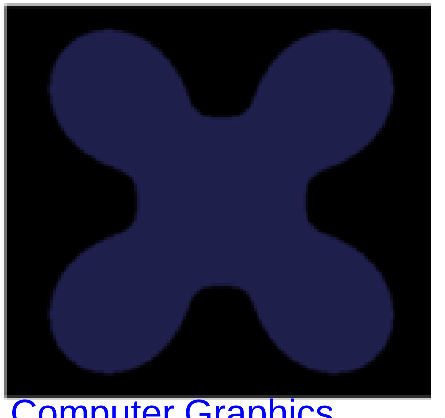
Images: http://xoax.net/comp/sci/graphics3D/BasLocIll.php

Ambient Light

ambient light

. light that doesn't come from any direction

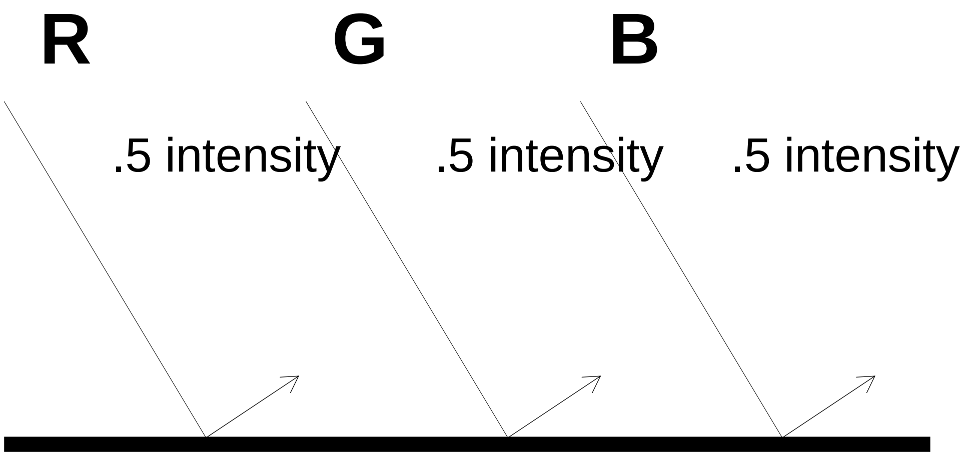
. objects are evenly lit on all surfaces in all directions



ambient light

. has a source, but rays of light bounce around the scene and become directionless

ambient light source

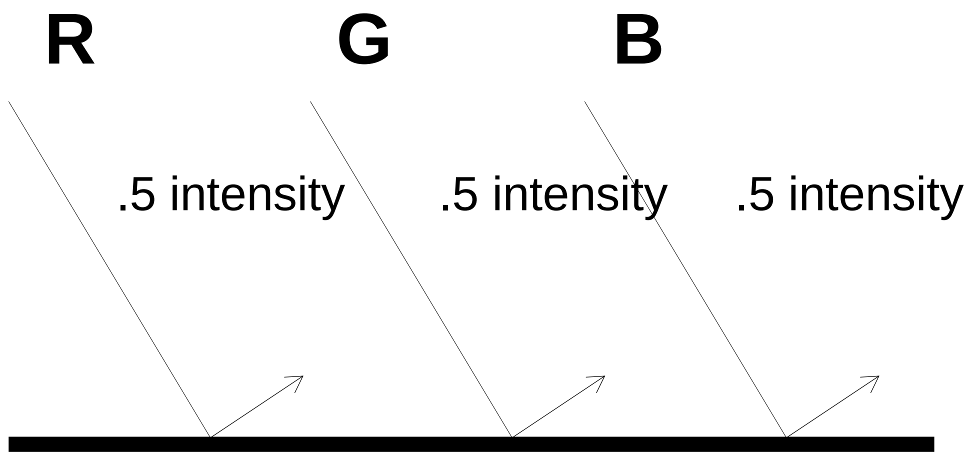


material "ambient" color (.5, 1, .5)

ambient light

. has a source, but rays of light bounce around the scene and become directionless

ambient light source



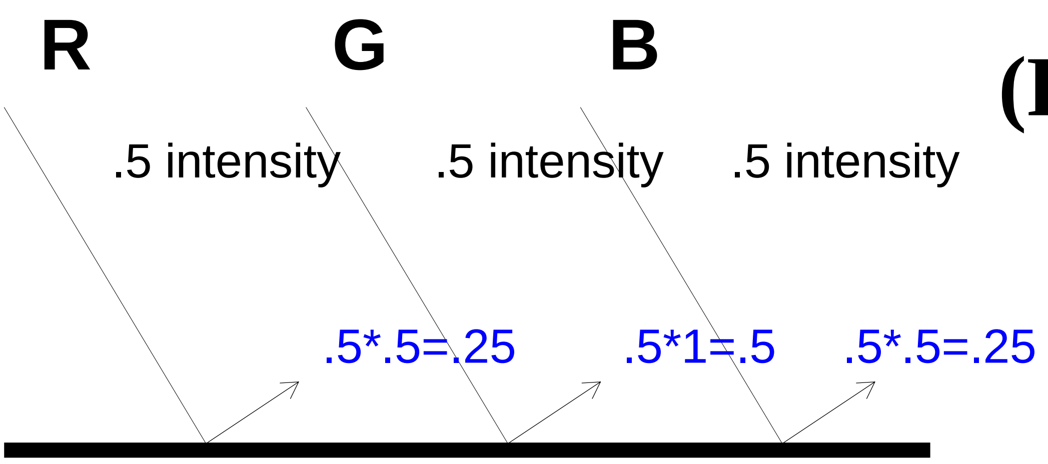
material "ambient" color (.5, 1, .5)

how do you calculate the ambient color component of an object?

ambient light

. has a source, but rays of light bounce around the scene and become directionless

ambient light source



material "ambient" color (.5, 1, .5)

color vector (R,G,B) = (.25, .5, .25)

Diffuse illumination

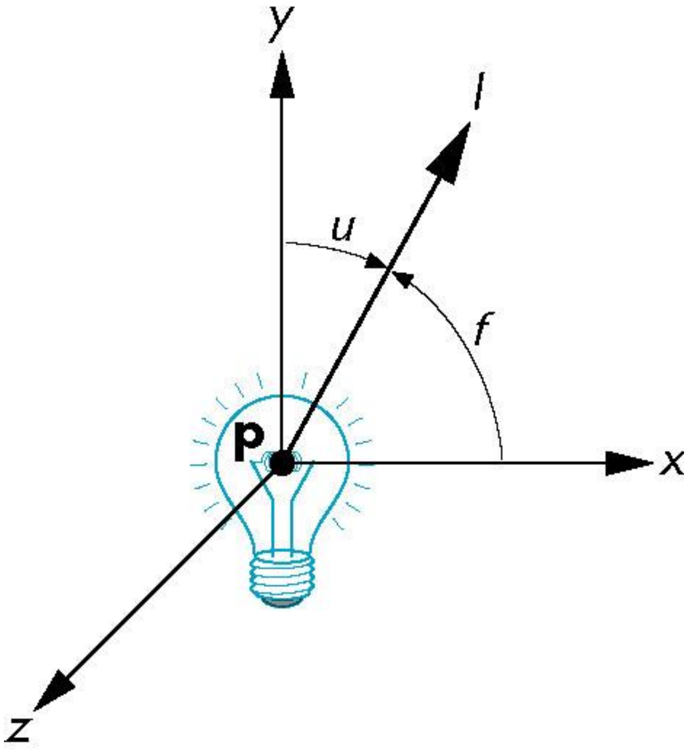
how can we create a light model?

$$\boldsymbol{I(x,y,z,\theta,\phi,\lambda)}$$

. *(x,y,z): light source*

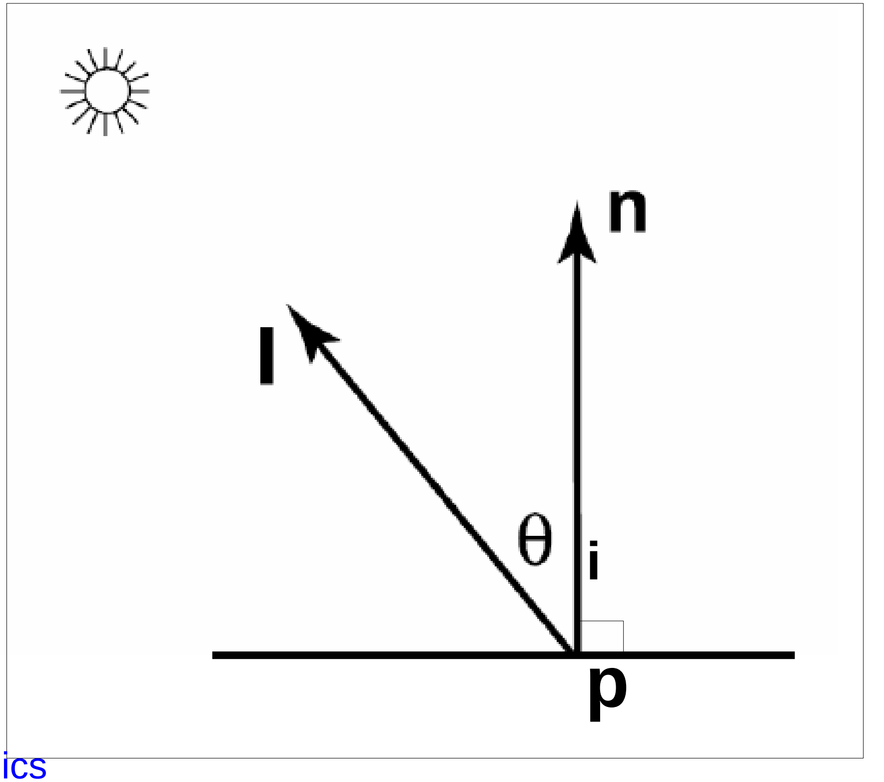
. *(θ,φ): emition direction*

. λ : light intensity



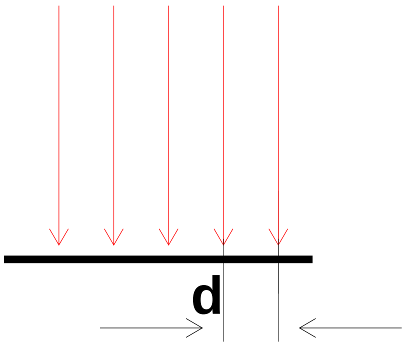
how can we create a light model?

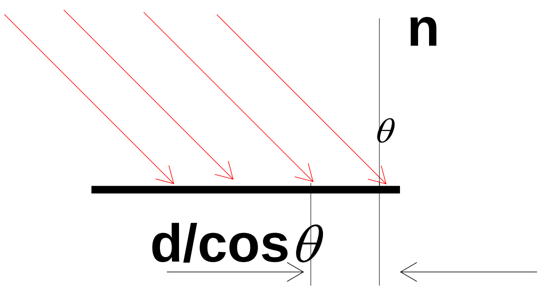
measuring **<u>irradiance</u>** at a **<u>plane perpendicular</u>** to **l** tells us how **<u>bright</u>** the light is in general



how can we create a light model?

meassures the density of the rays





Irradiance is **proportional** to the **density** of the rays

Inversely proportional to the distance **d** between the rays

Since **irradiance** is inversely proportional to the distance **d** it is **proportional** to **cos** θ

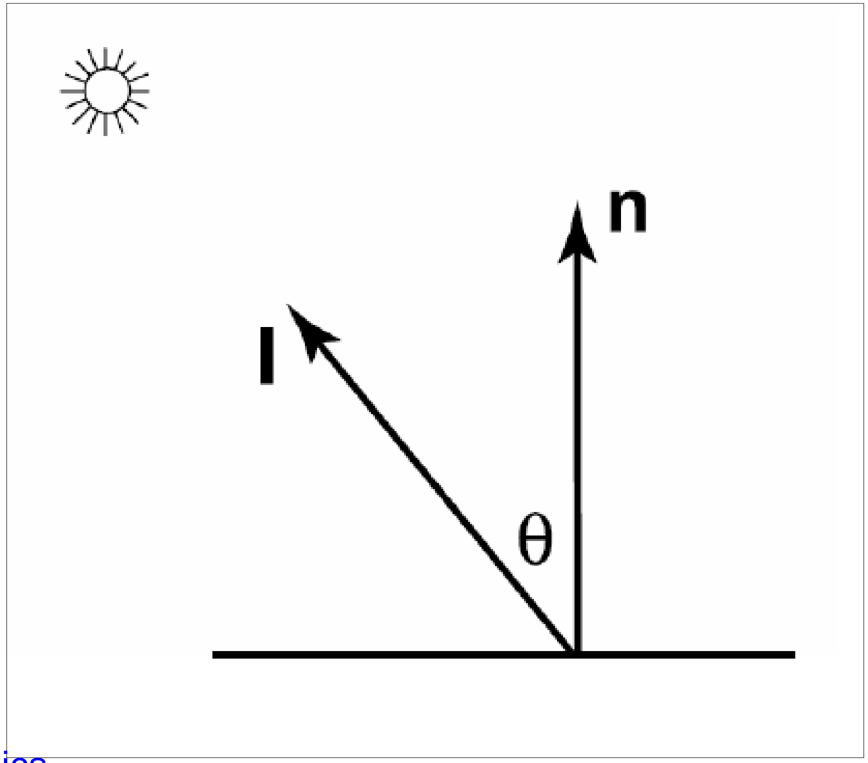
Lambert's law

$$I_{diffuse} = k_d \; I_{light} \cos(\theta)$$
$$= k_d \; I_{light} \, \boldsymbol{n} \cdot \boldsymbol{l}$$

I_{light}: light source intensity

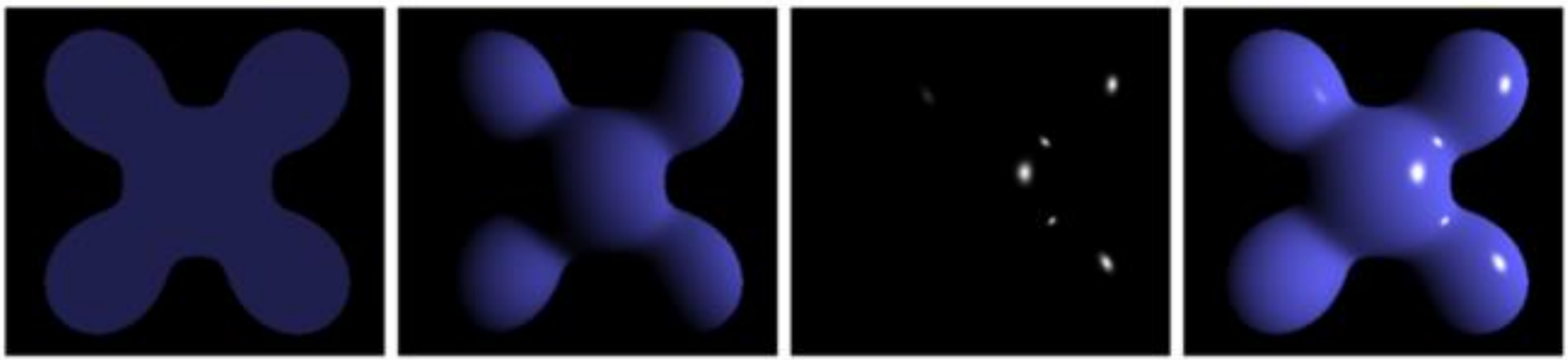
k : surface reflectance coefficient in [0,1]

θ: light/normal angle



Illumination: components

phong



ambient + diffuse + specular = phong

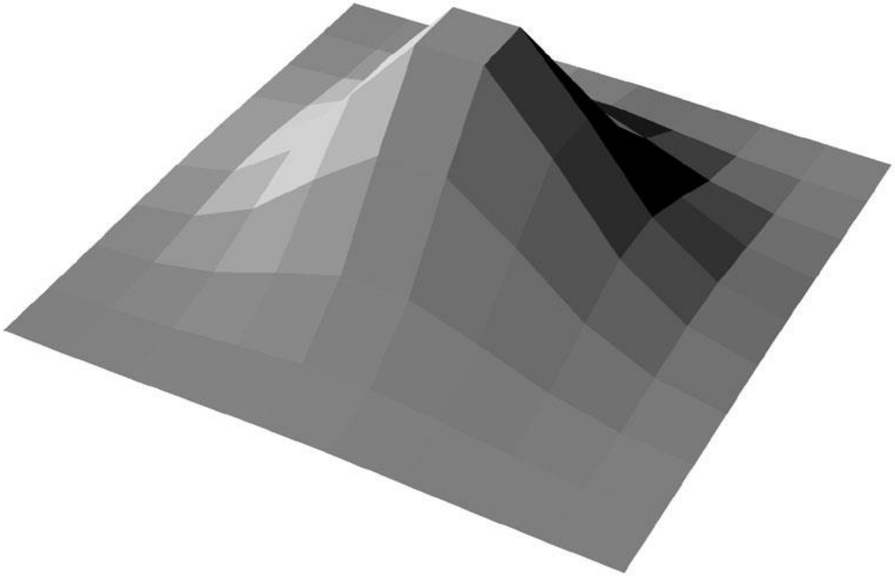
reflection

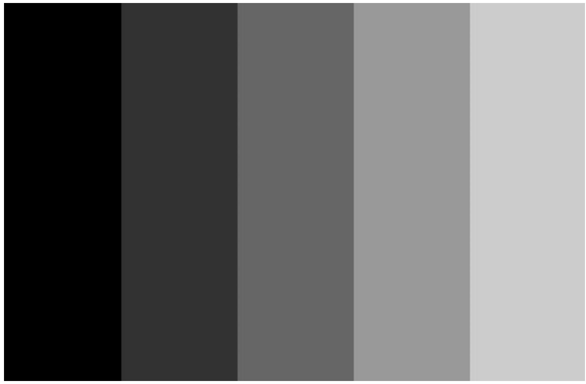
other: blinn-phong, lambert, gouraud,...

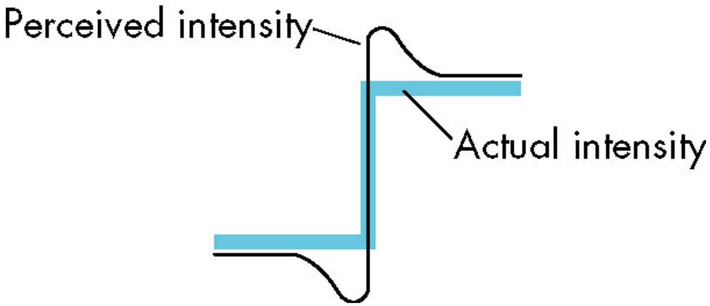
Shading

U.PORTO

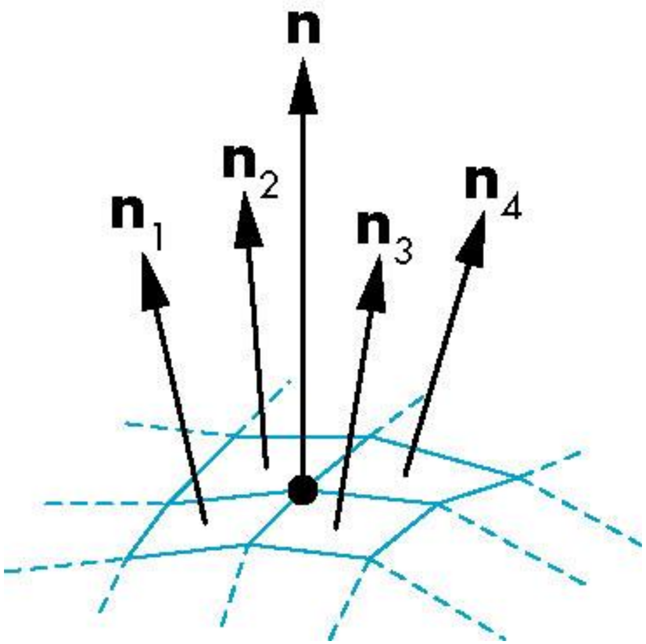
flat shading (ambient)







gouraud (smooth) shading

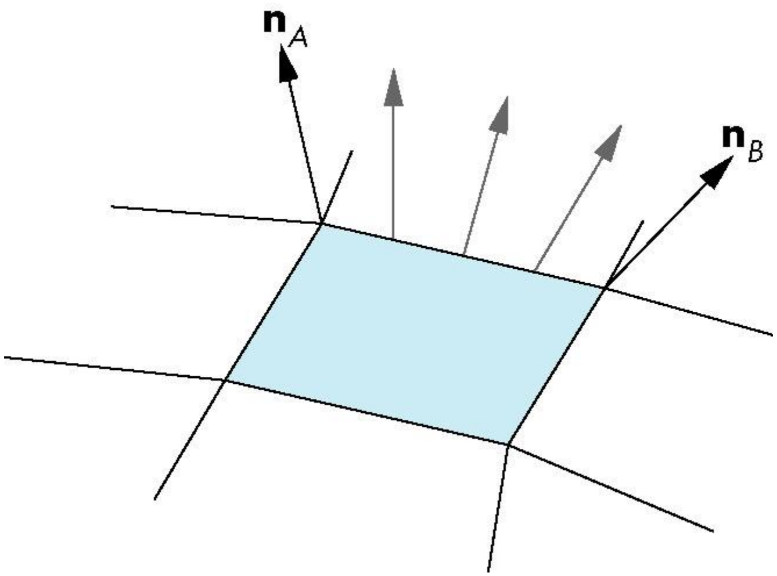


$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{\left|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4\right|}$$

– In OpenGL: `glShadeModel(GL_SMOOTH)`

phong (smooth) shading

1. calculate the normals on the side of the polygons by interpolating the vertex normals

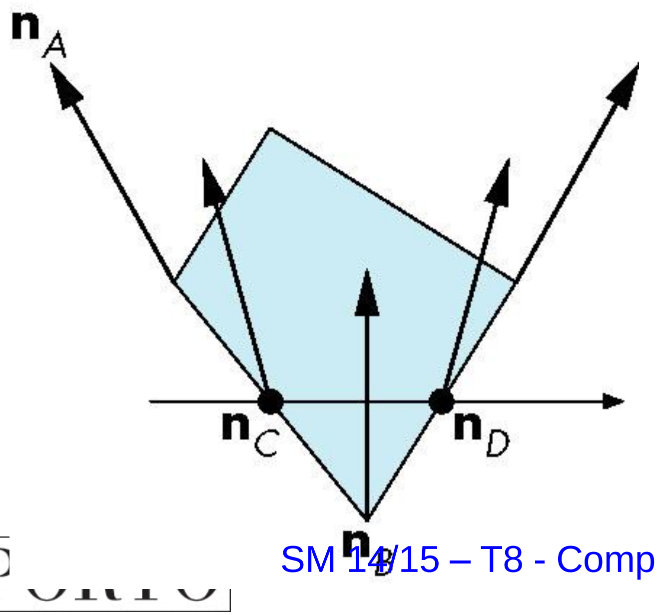


$$\mathbf{n}(\alpha) = (1-\alpha)\mathbf{n}_A + \alpha\mathbf{n}_B$$

phong (smooth) shading

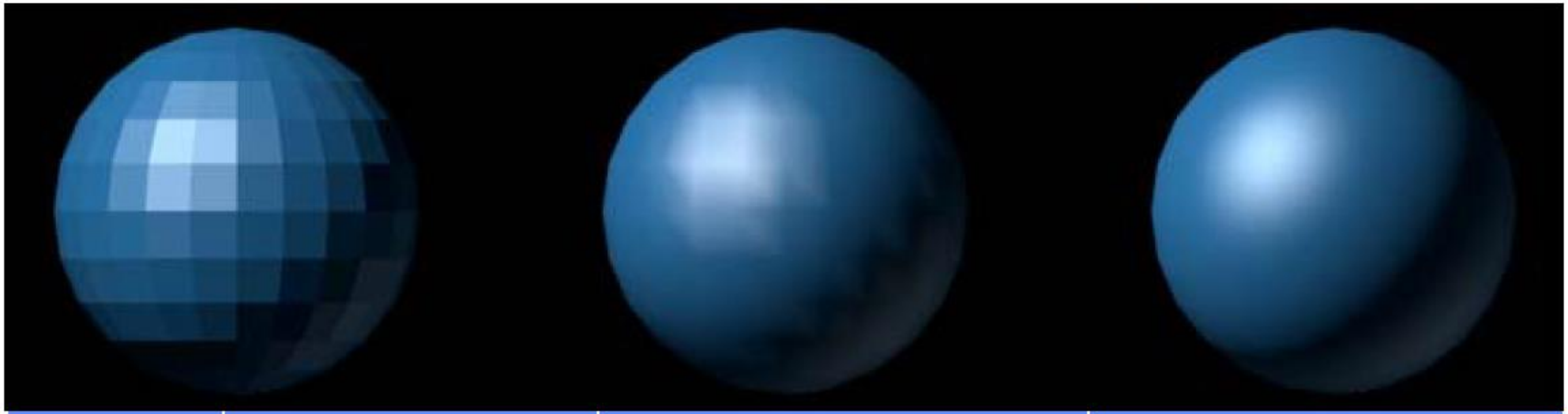
1. calculate the normals on the side of the polygons by interpolating the vertex normals

2. calculate the normals inside the polygon



$$\mathbf{n}(\alpha,\beta) = (1-\beta)\mathbf{n}_C + \beta\mathbf{n}_D$$

Flat / Gouraud / Phong Shading



	Flat Shading	Gouraud Shading	Phong Shading*
Normal	Split; same for each triangle's three vertices	Each vertex has a normal which is used to compute a per vertex color	Interpolated to each fragment
Color	One color value computed for each triangle	Interpolated to each fragment	Each fragment has a normal which is used to compute a per fragment color

Slide by Ron Fedkiw, Stanford University

In a nutshell

- Calculate each primary color separately
- Start with global ambient light
- Add reflections from each light source
- Clamp to [0, 1]

Summary: Illumination

- Three main types of light:
  - Ambient, Diffuse, Specular
- Illumination on a surface depends on the irradiance angle with the normal
  - Lambert shading model
- How can we calculate these normals?
  - Flat shading, Gouraud shading, Phong shading

Texture

introduction

textures are a way to
add detail to a surface






Images from http://www.cgtextures.com

how?

1. model the surface with more polygons
   . it is hard to model subtle details
   . more surface details, more rendering speed

2. map a texture to the surface
   . allows including more detail on the surface without affecting the rendering speed

Images from http://www.cgtextures.com

Particle Systems

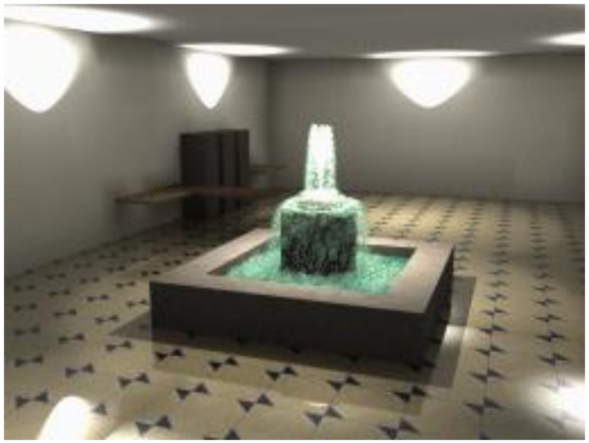
U.PORTO

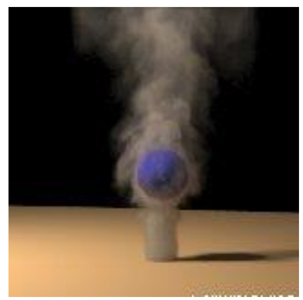
introduction

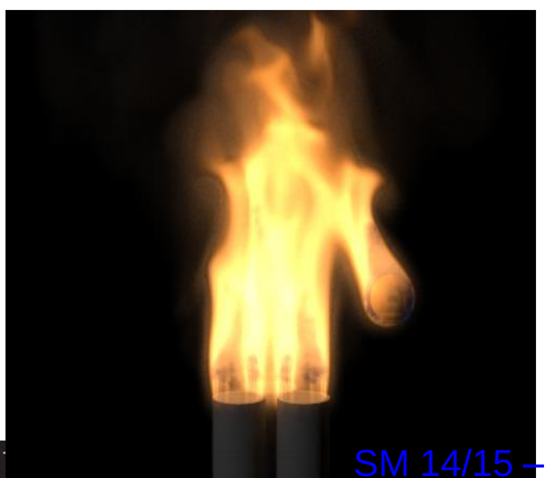
Particles systems **what for**?

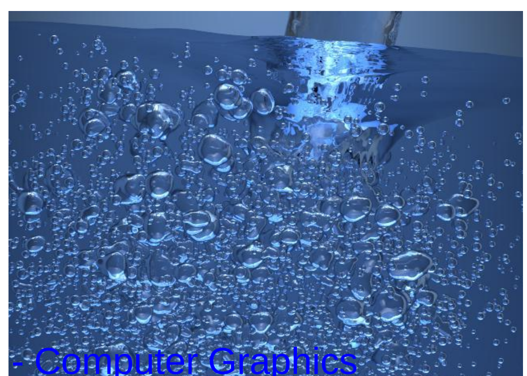
solution to modeling amorphous, dynamic and fluid objects like clouds, smoke, water, explosions and fire.

How can we do it?



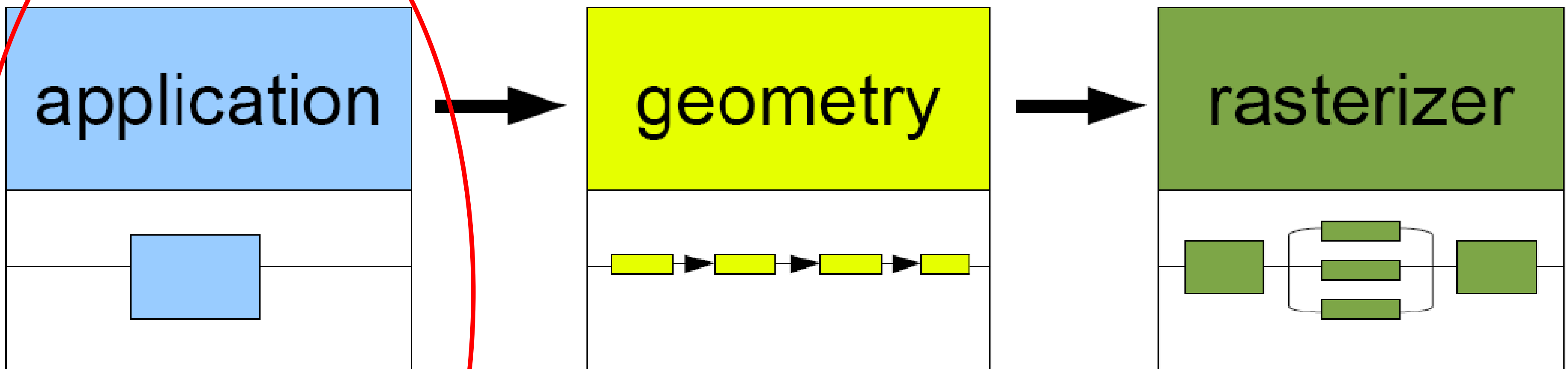






Ron Fedkiw
Jeong-Mo Hong

Pipeline



application

. **collision** detection
. **animation** global acceleration
. **physics** simulation

process on CPU or GPU

geometry

. **transformation**
. **projection**

Computes:
. what is to be drawn
. how should be drawn
. where should be drawn

process on GPU

rasterizer

. **draws** images generated by **geometry stage**

process on GPU

representing objects with particles

- An object is represented as **clouds of primitive particles** that define its volume rather than by polygons or patches that define its boundary

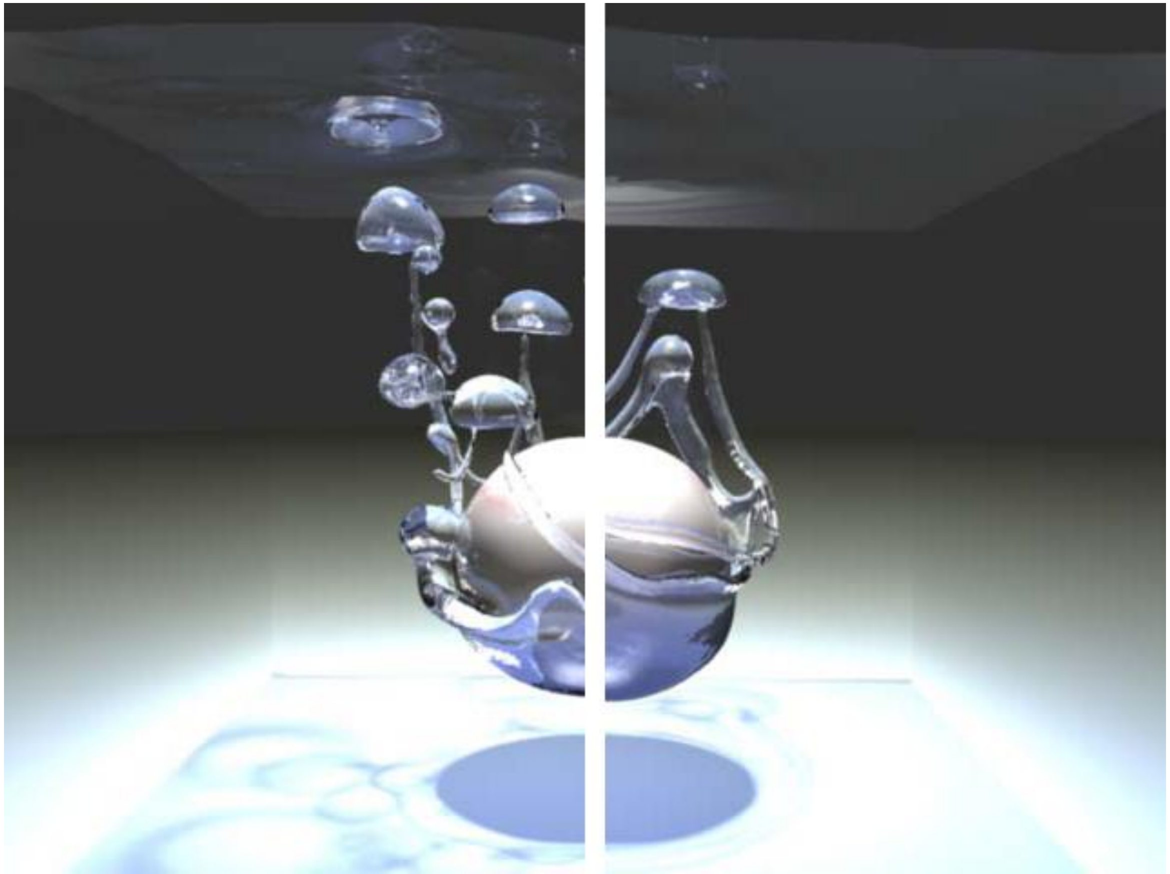
- A particle system is **dynamic**, particles changing form and moving with the passage of time.

- Object is **not deterministic**, its shape and form are not completely specified

Basic Model of Particle Systems

1) New particles are **generated** into the system
2) Each new particle is **assigned** its individual **attributes**
3) Any particle that has existed past its prescribed lifetime is **extinguished**
4) The **remaining** particles are **moved** and **transformed** according to their dynamic attributes
5) An image of the **particles** is **rendered** in the frame buffer, often using special purpose algorithms.

Particle rendering



Particles can obscure other objects behind them, can be transparent, and can cast shadows on other objects. The objects may be polygons, curved surfaces, or other particles.

Physics

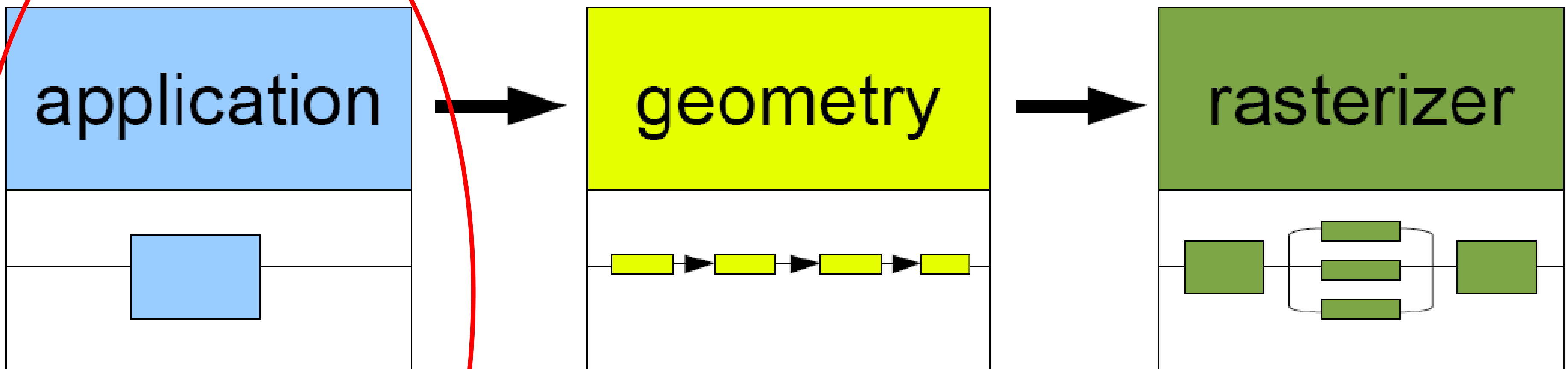
U. PORTO

Why do we need physics?

- How do objects move?
- How much energy do they have?
- How do they stop by themselves?
- How do they float?
- How do they fly?

And that only involves movement… (Heat? Electricity? Wind? Light? Sound?)

Pipeline



Application:
- . **collision** detection
- . **animation** global acceleration
- . **physics** simulation

process on CPU or GPU

Geometry:
- . **transformation**
- . **projection**

Computes:
- . what is to be drawn
- . how should be drawn
- . where should be drawn

process on GPU

Rasterizer:
- . **draws** images generated by **geometry stage**

process on GPU

Motion

The basic law

- Newton's second law $\vec{F} = m.\vec{a}$
- Force (N) equals mass (kg) times acceleration (ms^{-2})
- Means that accelerating an object requires an external force
- Also means that if we know this force, mass, and initial conditions we can predict object motion

Position and velocity

- If we know acceleration
- We can integrate it over time to obtain velocity
- And integrate it again to obtain position

We can predict motion!

$$\vec{v} = \int_t \vec{a}.dt$$

$$\vec{v} = \overrightarrow{v_0} + \vec{a}.t$$

$$\vec{x} = \int_t \vec{v}.dt$$

$$\vec{x} = \overrightarrow{x_0} + \overrightarrow{v_0}.t + {}^{1}/_{2}\,\vec{a}.t^2$$

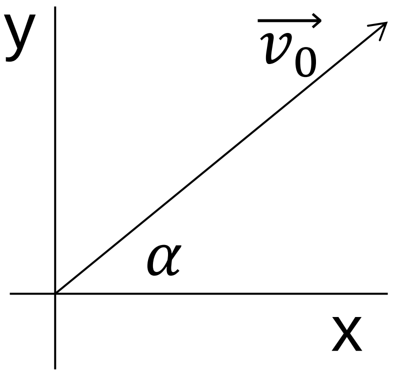
Vectors

- Note that position, acceleration and velocity are vectors
- Scalars are simpler…
- Use scalar versions of the equations for each dimension
  - x, y, z
- Separability makes things much simpler!

Example

- Break down the vector equation into its components *x* and *y*
- Use them independently
  - Great for calculating gravity effects of projectiles

$$\vec{v} = \overrightarrow{v_0} + \vec{a}.t$$



$$v_{0x} = |\overrightarrow{v_0}|.\cos\alpha$$
$$v_{0y} = |\overrightarrow{v_0}|.\sin\alpha$$

$$v_x(t) = v_{0x} + a_x(t).t$$
$$v_y(t) = v_{0y} + a_y(t).t$$

Projectile motion

- No force affects horizontal axis

  $a_x = 0$

- Gravity affects vertical axis

  $a_y = g = -9{,}8\ ms^{-2}$

- So:

$$x(t) = x_0 + v_{0x}.t$$

$$y(t) = y_0 + v_{0y}.t - {}^{1}/_{2}\,9{,}8.t^2$$

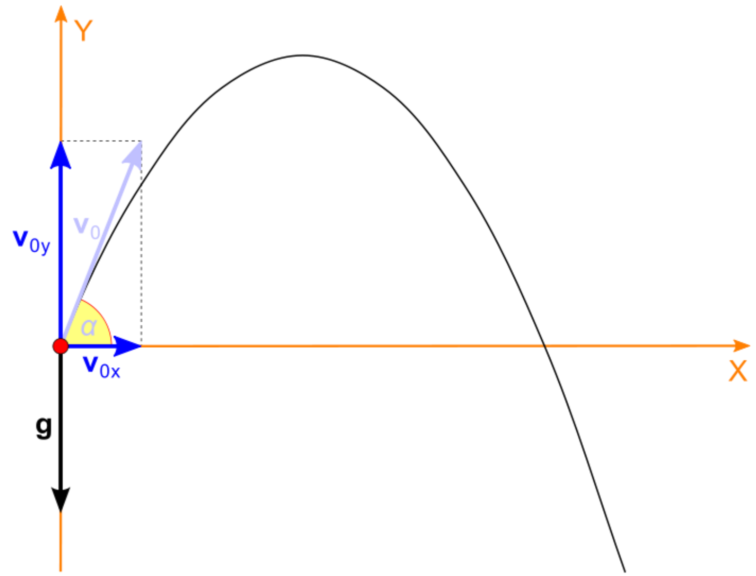


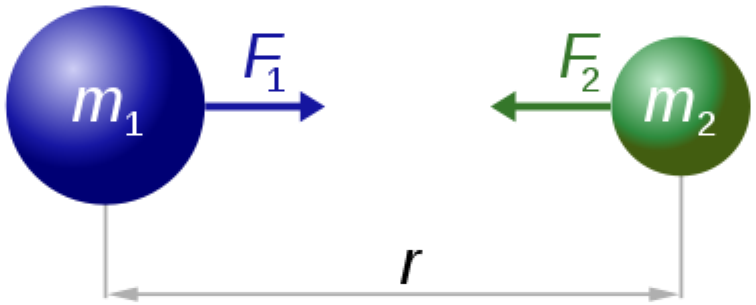
Image adapted from www.wikipedia.org

Engines

- How do I simulate an engine propelling an object?
  - I can use *force* if I know *mass*
  - I can use *acceleration* directly
- More difficult than gravity
  - Direction of acceleration is usually associated with the direction of velocity
  - Direction and magnitude of acceleration may be influenced externally: brakes, steering wheel, etc.
- Can easily combine with gravity

Gravitational force

- Any two objects with mass attract each other
  - Newton’s law of universal gravitation
- Direction of force
  - Line containing the *centers of mass* of the two objects
- How come earth’s gravitational pull is constant then?
  - It is not…



$$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$$

$G = 6.674 \times 10^{-11}\ \mathrm{N\ m^2\ kg^{-2}}$

Image adapted from www.wikipedia.org

Warp speed

- Near the speed of light
  - Mass increases with velocity
  - Mass deforms space
  - Things get messy
- What to do?
  - Go read Stephen Hawking
  - Cheat in your space combat simulation

Energy of moving objects

Kinetic energy

- Things in motion have *energy*
  - Defined as the *work* needed to accelerate a body of a given *mass* from rest to its stated *velocity*
  - Measured in *joules*
- Classic mechanics
  - Kinetic energy of a non-rotating rigid body:

$$E_k = {}^{1}/_{2}\, mv^2$$

Potential energy

- Things not moving also have ‘potential’ energy
  - Energy due to the position of the various objects of a system
- Most common potential energy
  - Gravity $E_p = m.g.h$
  - Higher objects have higher energy than lower objects with the same mass
  - Others: elastic, electric, magnetic

Conservation of energy

- Law of conservation of energy
  - The total amount of *energy* in an *isolated system* remains constant over time
- Isolated system
  - Physical system without any external exchange of matter or energy
- Great for approximating many real-world situations!

Back to our projectiles

- Projectile going up
  - Velocity decreasing – lower kinetic energy
  - Height increasing – higher potential energy
- Projectile going down
  - Vice-versa
- What about engines?
  - External energy source
  - Not an isolated system!

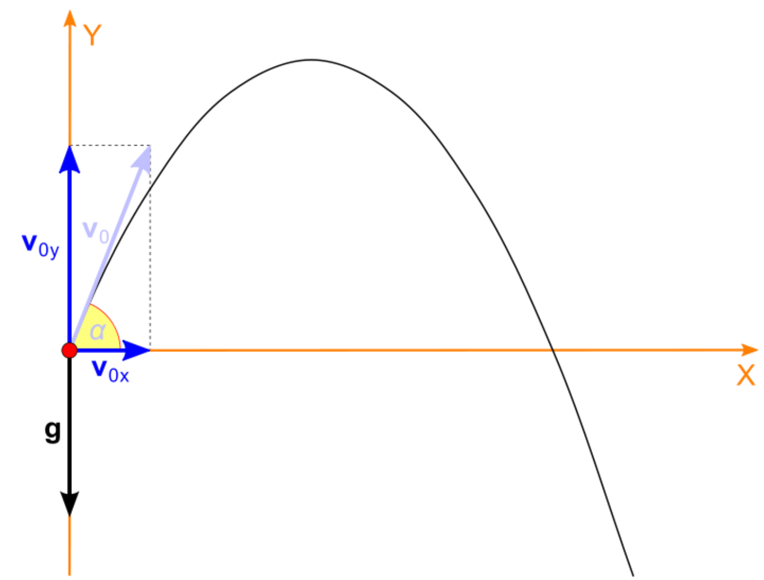


Image adapted from www.wikipedia.org

Object collision

- What happens when my projectile falls to the ground?
  - Law of conservation of energy
  - No external forces were applied
  - What happened to the kinetic energy?
- Ground must generate an opposing force that stops the projectile
  - Which could break or deform the ground…
- Energy is typically converted into heat
  - Explaining why even a small asteroid falling on earth can create a huge explosion…

Momentum

- What happens when two objects collide?
  - All collisions conserve *momentum*
  - Not all collisions conserve *kinetic energy*
- What is *momentum*?

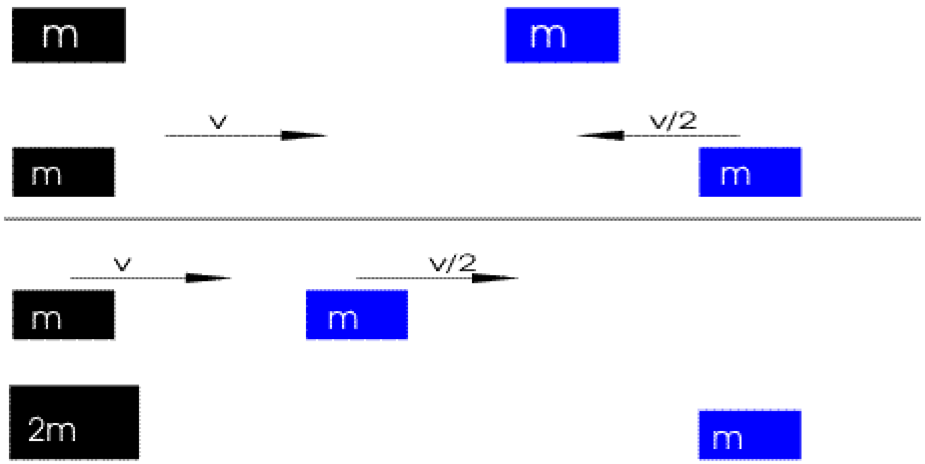
$$p = m.v$$

  - Product of *mass* and *velocity* of an object
  - *Conserved* in a *closed system*
- The momentum of a system of particles is the sum of their momenta

$$p = p_1 + p_2 = m_1 v_1 + m_2 v_2$$

Elastic collisions

- *Momentum* is conserved
- *Total kinetic energy* is conserved
- Solvable system of equations



$$m_1u_1 + m_2u_2 = m_1v_1 + m_2v_2$$
$$\tfrac{1}{2}m_1u_1^2 + \tfrac{1}{2}m_2u_2^2 = \tfrac{1}{2}m_1v_1^2 + \tfrac{1}{2}m_2v_2^2\,.$$

Images adapted from www.wikipedia.org

Inelastic collision

- *Momentum* is conserved
- *Kinetic energy* is not conserved
- Coefficient of restitution
  - Fractional value representing the ratio of speeds after and before an impact

Images adapted from www.wikipedia.org

Why do moving objects stop?

(without collisions or brakes…)

U. PORTO

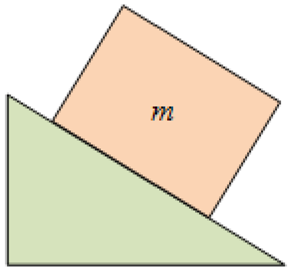
Reason #1 - Friction

- Force resisting the relative motion of solid surfaces in contact
  – Actually this is *dry kinetic friction*…
- Coulomb friction

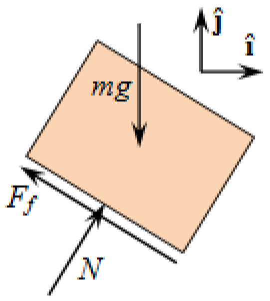
$$F_f \leq \mu F_n$$

- Does not depend on velocity!
- Depends on the *normal force* between two surfaces

A block on a ramp



Free body diagram
of just the block



Images adapted from www.wikipedia.org

Reason #2 - Drag

- Forces which act on a solid object in the direction of the relative fluid flow
  - Air resistance
  - Fluid resistance
- Depends on velocity and the object's cross-sectional area

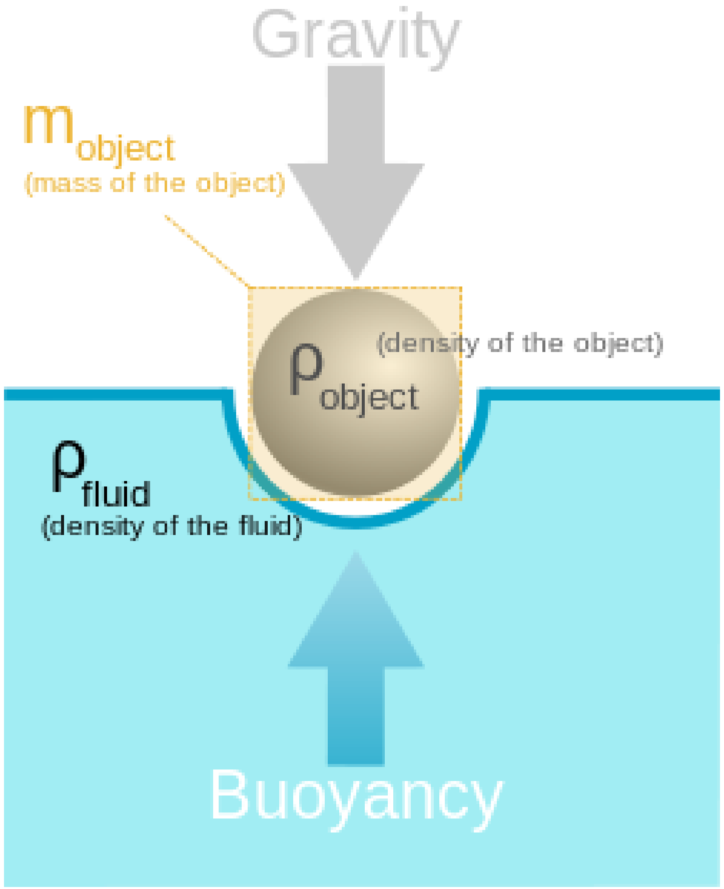
$$F_D = \frac{1}{2} \rho v^2 C_D A$$

- More complex than friction
- Use simple models (Stokes', Newton…)

Why do things float?

Buoyancy

- Archimedes' principle
  - A body immersed in a fluid suffers an upward force equal to the weight of the fluid the body displaces
- Objects float if they are less dense than the fluid they are in
  - Can you model such an object falling on a fluid?



Images adapted from www.wikipedia.org

How do explosions work?
How can I model turbulence?
How do things fly?
Why do cars get lighter as they go faster?

...

U. PORTO

Go read about physics!

Physics in space? ☺

U.PORTO

http://youtu.be/o8TssbmY-GM