Image Processing and Computer Graphics

Riccardo Salvalaggio

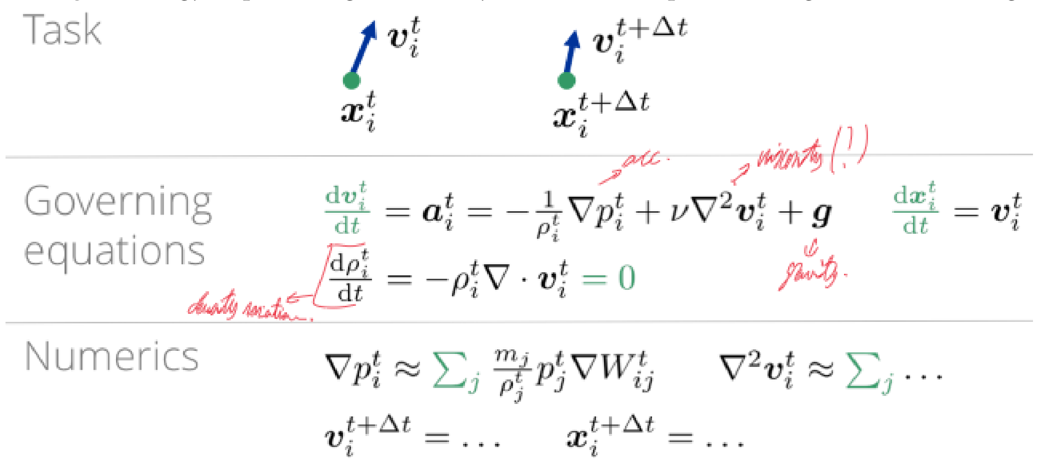
19th of April, 2021

Contents

1 Introduction Computer Graphics

Modeling: generate, represent geometry.
Rendering: light transposing, delete objects etc.
Simulation: animation, dynamic representation.

Light: energy or photons generated by a source, transported along lines, interacting at surfaces (reflection).



Task

v_i^t
x_i^t
$v_i^{t+\Delta t}$
$x_i^{t+\Delta t}$

Governing equations

$$\frac{dv_i^t}{dt} = a_i^t = \overset{\text{acc.}}{-\frac{1}{\rho_i^t}\nabla p_i^t} + \overset{\text{viscosity (!)}}{\nu\nabla^2 v_i^t} + \underset{\text{gravity.}}{g} \qquad \frac{dx_i^t}{dt} = v_i^t$$

$$\underset{\text{density variation.}}{\boxed{\frac{d\rho_i^t}{dt}}} = -\rho_i^t \nabla \cdot v_i^t = 0$$

Numerics

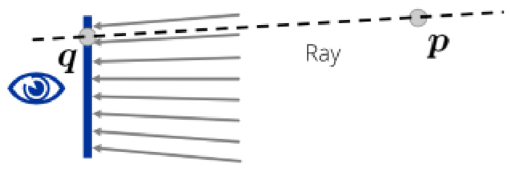
$$\nabla p_i^t \approx \sum_j \frac{m_j}{\rho_j^t} p_j^t \nabla W_{ij}^t \qquad \nabla^2 v_i^t \approx \sum_j \dots$$

$$v_i^{t+\Delta t} = \dots \qquad x_i^{t+\Delta t} = \dots$$

Pressure is computed by solving a pressure Poisson equation.

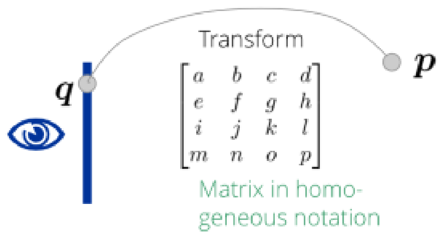
1.1 Rendering aspects

- Ray Tracing:



Ray Tracers compute ray-scene intersections to estimate q from p.

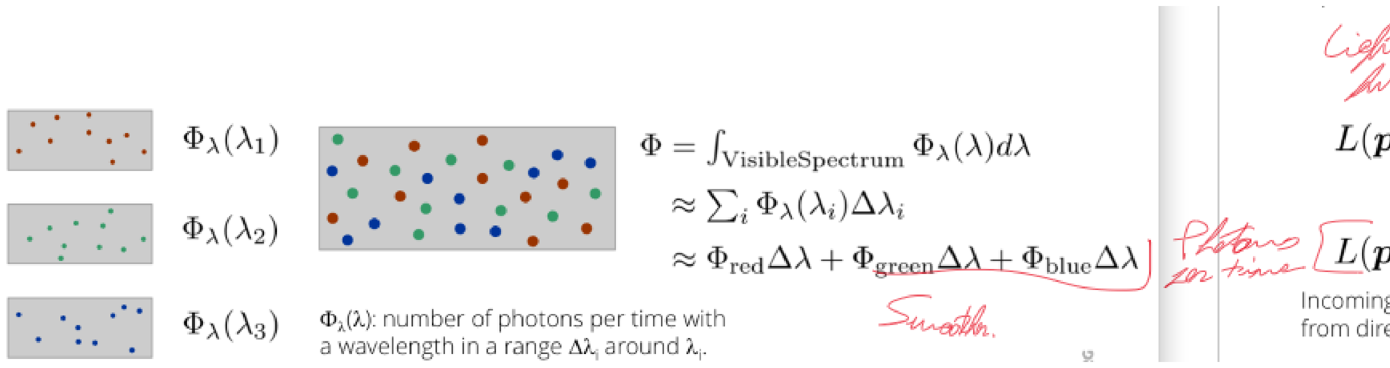
- Rasterization:


Transform
q
p
$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$
Matrix in homogeneous notation

Rasterizers apply transformations to p in order to estimate q. p is projected onto the sensor plane.

1.2 Light

Photons are characterized by a wavelength within the visible spectrum =¿ color.


$\Phi_\lambda(\lambda_1)$

$\Phi_\lambda(\lambda_2)$

$\Phi_\lambda(\lambda_3)$

$$\Phi = \int_{\text{VisibleSpectrum}} \Phi_\lambda(\lambda) d\lambda$$
$$\approx \textstyle\sum_i \Phi_\lambda(\lambda_i)\Delta\lambda_i$$
$$\approx \Phi_{\text{red}}\Delta\lambda + \Phi_{\text{green}}\Delta\lambda + \Phi_{\text{blue}}\Delta\lambda$$

Smooth.

Photons per time

$\Phi_\lambda(\lambda)$: number of photons per time with a wavelength in a range $\Delta\lambda_i$ around λ_i.

$L(p$

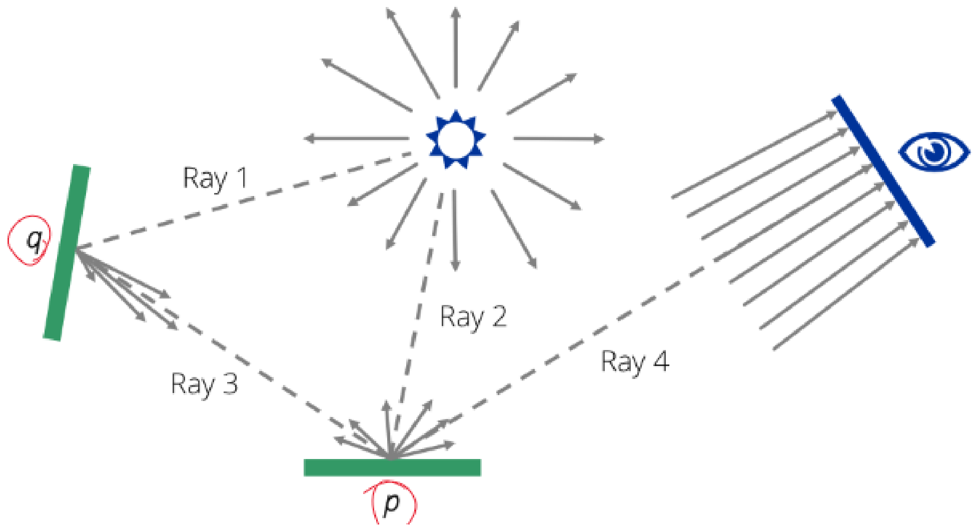
$L(p$

Incoming
from dire

Rendering -¿ lookup light transported along rays casted into the scene.

2 Ray Casting

Rendering problem: visibility/hidden surface problem, object projection onto sensor plane. (Rat-object intersections with ray casting/tracing).



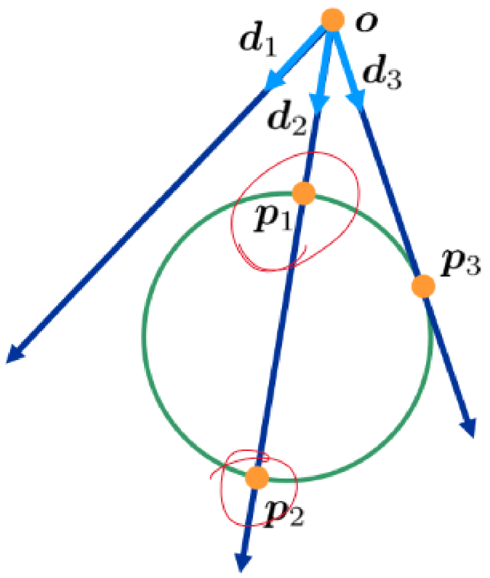
Goal: to compute ray 4 (incoming light at the sensor considering every riflection).
Increasing the number of rays we are considering, we obtain an higher precision (great computationally effort).
Primary rays: start/end at sensors
Secondary rays: don't start/end at sensors
Shadow rays: start/end at sources
Primary solve visibility problem, secondary are useful in order to compute light transport.
Ray casting: computation of position p (what is visible).
Ray tracing: computaiton of light transportd (which colors).
Rendering equations to compue incoming light. **Concept:** a ray is a half-line specified by an origin o and a direction d =¿ r(t) = o+td with t¿=0. We have to compute nearest intersection with all objects.

2.1 Implicit surfaces

If p = x,y,z is a surface point =¿ f(x,y,z) = 0.
Intersection: f(x,y,z) = f(r(t)) = f(o+td) = 0. All point on a normal surface with offset r satisfy that intersection =¿ n*(p-r) = 0. If d is not orthogonal to n, the intersection can be computed: n*(o+td-r)=0, t=[(r-o)*n]/[n*d].
n is given by gradient (function that compute the variation for unit) of the implicit function (n is the direction with the greater value of gradient).

$$\boldsymbol{n} = \nabla f(\boldsymbol{p}) = \left(\frac{\partial}{\partial x} n_x(x - r_x), \frac{\partial}{\partial y} n_y(y - r_y), \frac{\partial}{\partial z} n_z(z - r_z)\right) = (n_x, n_y, n_z)$$

For quadrics we use the same knowledge expanded to three dimensions (quadratic equations).



Ray 1: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}_1$

$$B^2 - 4AC < 0$$

Ray 2: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}_2$

$$t_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

$$\boldsymbol{p}_{1,2} = \boldsymbol{o} + t_{1,2}\boldsymbol{d}_2$$

Ray 3: $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}_3$

$$t_3 = \frac{-B}{2A}$$

$$\boldsymbol{p}_3 = \boldsymbol{o} + t_3\boldsymbol{d}_3$$

2.2 Parametric surfaces

Represented by functions with 2D parameters: x = f(u,v), y = g(u,v), z = h(u,v)
Intersetion can be computer from a non-linear system.

$$o_x + td_x = f(u,v) \quad o_y + td_y = g(u,v) \quad o_z + td_z = h(u,v)$$

Normal vector

$$\boldsymbol{n}(u,v) = \underbrace{\left(\frac{\partial f}{\partial u}, \frac{\partial g}{\partial u}, \frac{\partial h}{\partial u}\right)}_{\text{Tangent}} \times \underbrace{\left(\frac{\partial f}{\partial v}, \frac{\partial g}{\partial v}, \frac{\partial h}{\partial v}\right)}_{\text{Tangent}}$$

Parametric representations are used to render partial objects.

2.3 Combined objects

Compound objects: union of forms.
Constructive Solid Geomtry CSG: combine simple objects to complex geometry using boolean operators. Estimate and analyze all intersections considering intervals inside objects, works for closed surfaces.

2.4 Triangles

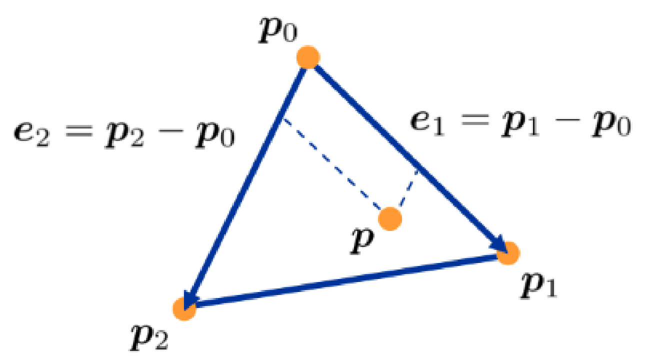
– Parametric representation (Barycentr

$$p(b_1, b_2) = (1 - b_1 - b_2)p_0 + b_1 p_1 + b_2 p_2$$

$$b_1 \geq 0 \qquad b_2 \geq 0 \qquad b_1 + b_2 \leq 1$$

Verti
p is a
of th

Contraints to be
Can also [illegible] in



$$\begin{aligned} p &= p_0 + b_1 \\ &= p_0 + b_1 \\ &= \underbrace{(1 - b_1}_{b_0} \end{aligned}$$

Popular appoximate surface representation
Intersection: o+td = (1-b1-b2)p0 + b1p1 + b2p2.

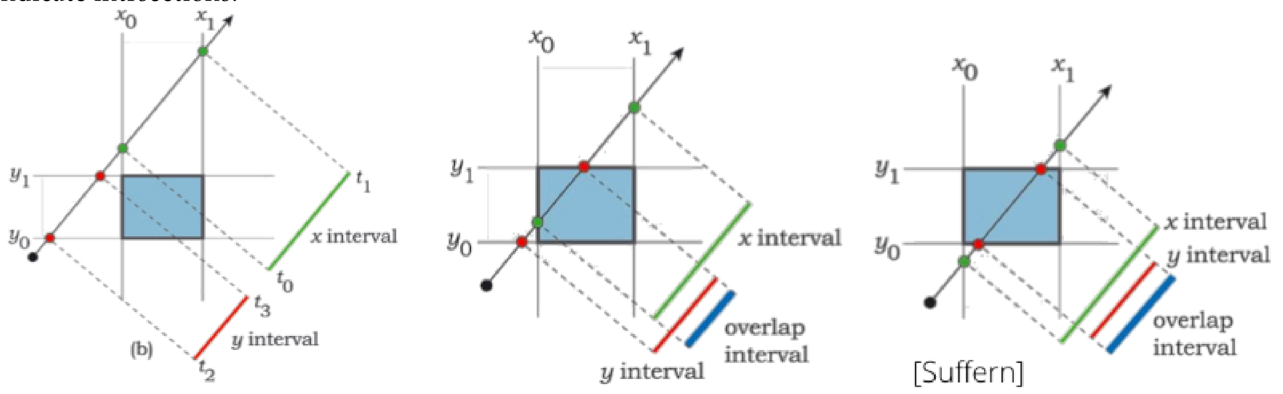
Solution

$$\begin{pmatrix} t \\ b_1 \\ b_2 \end{pmatrix} = \frac{1}{(d \times e_2) \cdot e_1} \begin{pmatrix} (s \times e_1) \cdot e_2 \\ (d \times e_2) \cdot s \\ (s \times e_1) \cdot d \end{pmatrix}$$

2.5 Axis-aligned boxes

2.5.1 Axis-Aligned Bounding Box (AABB)

Simple geometry outside complex geometry, if a ray doesn't intersecate to the simple, surely doesn't intersect the complex. Boxes are represented by slabs. Intersections of rays are analyzed in order to check for ray-object intersection. Implementations is again done using normal intersection equation. Overlapping ray intervals indicate intrsections.



2.5.2 Bounding Volume Hierarchies (BVH)

AABBs combined in a hierarchy way. put smaller boxes recursively. Efficient pruning but memory and pre-processing overhead.

2.6 Iso-surfaces in grids

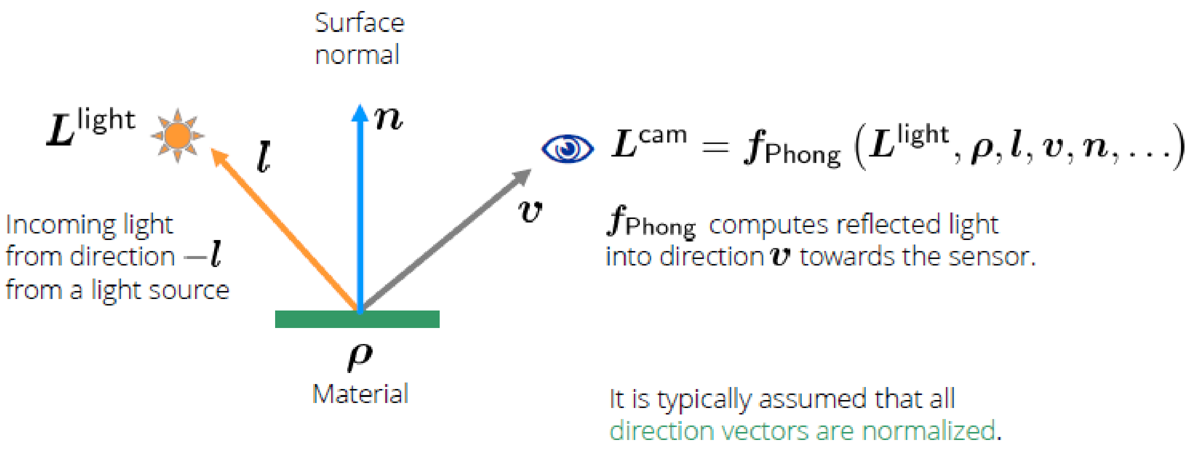
In order to compute intersections on fluid surfaces. Computation is based on density comparison.

Ray casting is very versatile concept to compute what is visible at a sensor but so expensive for complex geometries.

3 Shading

The way to compute color and intensity of what is visible by the sensor. Light is emitted by light sources but is scattered and or absorbed by surface (if dark aborbed, if bright reflected)and media.
Radiance: photons per time * area * solid angle (describe how much light is transported). **Coloured light** is represented by a 3D vector L: (Lred, Lgreen, Lblue). **Coloured objects** are characterized by reflectance coefficient ρ = (ρred, ρgreen, ρblue). Colours that are in the vector are reflected, otherwise absorbed.
We'll use Local Illumination Models that approximately solve the rendering equation.

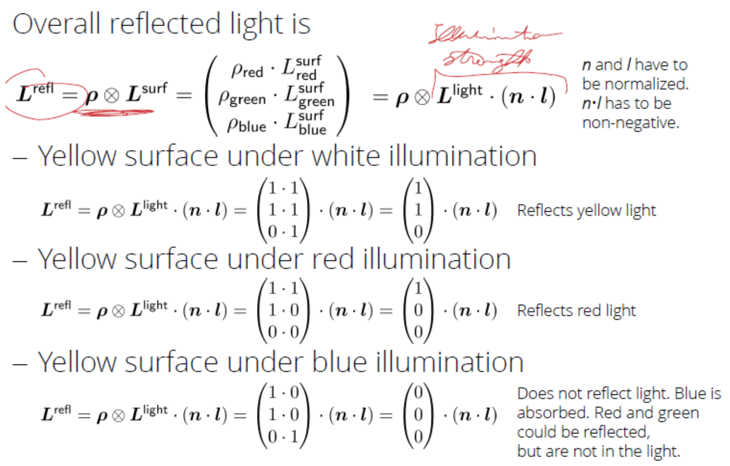
3.1 Phong Illumination Model



Surface normal
$\boldsymbol{L}^{\text{light}}$
$\boldsymbol{l}$
$\boldsymbol{n}$
$\boldsymbol{L}^{\text{cam}} = \boldsymbol{f}_{\text{Phong}}\left(\boldsymbol{L}^{\text{light}}, \boldsymbol{\rho}, \boldsymbol{l}, \boldsymbol{v}, \boldsymbol{n}, \ldots\right)$
$\boldsymbol{v}$
Incoming light from direction $-\boldsymbol{l}$ from a light source
$\boldsymbol{f}_{\text{Phong}}$ computes reflected light into direction $\boldsymbol{v}$ towards the sensor.
$\boldsymbol{\rho}$
Material
It is typically assumed that all direction vectors are normalized.

Lsurf is the surface illumination caused by Llight. Lrefl will depend on the object color ρ. Lcam is a portion of Lrefl transported to the sensor (will depend on the material).

Lambert's Cosine Law 1 *Illumination strenght at a surface is proportional to the cosine of the angle between l and n.*



Overall reflected light is

$$\boldsymbol{L}^{\text{refl}} = \boldsymbol{\rho} \otimes \boldsymbol{L}^{\text{surf}} = \begin{pmatrix} \rho_{\text{red}} \cdot L^{\text{surf}}_{\text{red}} \\ \rho_{\text{green}} \cdot L^{\text{surf}}_{\text{green}} \\ \rho_{\text{blue}} \cdot L^{\text{surf}}_{\text{blue}} \end{pmatrix} = \boldsymbol{\rho} \otimes \overbrace{\boldsymbol{L}^{\text{light}} \cdot (\boldsymbol{n} \cdot \boldsymbol{l})}^{\text{Illumination strength}}$$

n and *l* have to be normalized. *n·l* has to be non-negative.

– Yellow surface under white illumination

$$\boldsymbol{L}^{\text{refl}} = \boldsymbol{\rho} \otimes \boldsymbol{L}^{\text{light}} \cdot (\boldsymbol{n} \cdot \boldsymbol{l}) = \begin{pmatrix} 1 \cdot 1 \\ 1 \cdot 1 \\ 0 \cdot 1 \end{pmatrix} \cdot (\boldsymbol{n} \cdot \boldsymbol{l}) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \cdot (\boldsymbol{n} \cdot \boldsymbol{l})$$

Reflects yellow light

– Yellow surface under red illumination

$$\boldsymbol{L}^{\text{refl}} = \boldsymbol{\rho} \otimes \boldsymbol{L}^{\text{light}} \cdot (\boldsymbol{n} \cdot \boldsymbol{l}) = \begin{pmatrix} 1 \cdot 1 \\ 1 \cdot 0 \\ 0 \cdot 0 \end{pmatrix} \cdot (\boldsymbol{n} \cdot \boldsymbol{l}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot (\boldsymbol{n} \cdot \boldsymbol{l})$$

Reflects red light

– Yellow surface under blue illumination

$$\boldsymbol{L}^{\text{refl}} = \boldsymbol{\rho} \otimes \boldsymbol{L}^{\text{light}} \cdot (\boldsymbol{n} \cdot \boldsymbol{l}) = \begin{pmatrix} 1 \cdot 0 \\ 1 \cdot 0 \\ 0 \cdot 1 \end{pmatrix} \cdot (\boldsymbol{n} \cdot \boldsymbol{l}) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \cdot (\boldsymbol{n} \cdot \boldsymbol{l})$$

Does not reflect light. Blue is absorbed. Red and green could be reflected, but are not in the light.

Depending on the material, the reflection will be either Diffuse (if Matte) or Specular (if Shiny).

$$L^{\text{cam}} = \alpha \cdot \rho \otimes L^{\text{indirect}} + \sum_i L_i^{\text{light}} \cdot (n \cdot l_i) \otimes \left(\beta \cdot \rho + \gamma \cdot \rho^{\text{white}} \cdot (r_i \cdot v)^m\right)$$

Figure 1: Total model (coefficients are user-defined)

3.1.1 Diffuse Reflection

Matte surfaces reflect light equally into all directions.

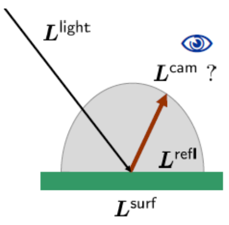
$$L^{\text{refl}} = \underline{\int_{2\pi} L(\omega_o) \cos\theta_o \mathrm{d}\omega_o}$$
Overall reflected light equals reflected light into a direction integrated over all directions

direction cos integration all directions 2π.

$$L(\omega_o) = \text{const} = L^{\text{cam}}$$
Definition of diffuse reflection

$$L^{\text{refl}} = L^{\text{cam}} \int_{2\pi} \cos\theta_o \mathrm{d}\omega_o = L^{\text{cam}} \cdot \pi$$

$$L^{\text{cam}} = \frac{1}{\pi} L^{\text{refl}} = \frac{1}{\pi} \cdot \rho \otimes L^{\text{light}} \cdot (n \cdot l)$$



3.1.2 Specular Reflection

Shiny surfaces reflect light into a small set of dominant directions.

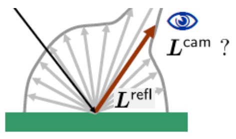
– Light L^{cam} towards sensor

$$L^{\text{refl}} = \int_{2\pi} L(\omega_o) \cos\theta_o \mathrm{d}\omega_o$$
Overall reflected light equals reflected light into a direction integrated over all directions

$$L(\omega_o) \sim (r \cdot \omega_o)^m$$
Definition of specular reflection. r is the reflection vector of light direction l with respect to normal n.

$$L^{\text{refl}} = \int_{2\pi} k(r \cdot \omega_o)^m \cos\theta_o \mathrm{d}\omega_o$$

$$L^{\text{cam}} = \rho^{\text{white}} \otimes L^{\text{light}} \cdot (n \cdot l) \cdot (r \cdot v)^m$$
k is not analyzed in Phong's model. White surface color accounts for the fact that shiny surfaces reflect the entire light spectrum.



Exponent m governs the size of the highlight area. M does not influence the maximal intensity. r can be computed as: r = 2*(l*n)*n-l.
According to the Blinn-Phong (Lrefl from a shiny surface) illumination model r = (l+v)/norma(l+v).

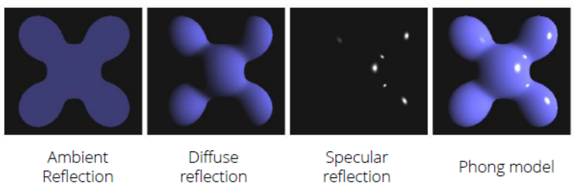
3.1.3 Reflection from Ambient Illumination

Indirect illumination from other surfaces. Reflecte light is: p_xorLindirect.

E.g., red cube in an illuminated room with yellow walls:

$$L^{\text{indirect}} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \rho^{\text{cube}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$L^{\text{cam}} = \frac{1}{\pi} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\pi} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$



Lcam = 1/_pi*p_xorLindirect.
In conclusion Phong is an efficient local computing model and its implementation can be parallelized.Drawbacks: resulting images tend to look less realistic because realistic scenes have more complex illuminations, complex material and non-physical phong parameters cause issues.

3.2 Extensions

Distances must be considered:
1. Between object surface and light source.
- Inverse Square Law: Illumination of a surface decreases quadratically with the distance from a light source =>Lsurf = $(1/r^2) * Llight * (n * l)$.

2. Between object surface and viewer.
- Fog: It is approximated by a combination of Lcam and Cfog. Lcam,fog = f(d)*Lcam+(1-f(d))*Cfog. f(d) describes the visibility.

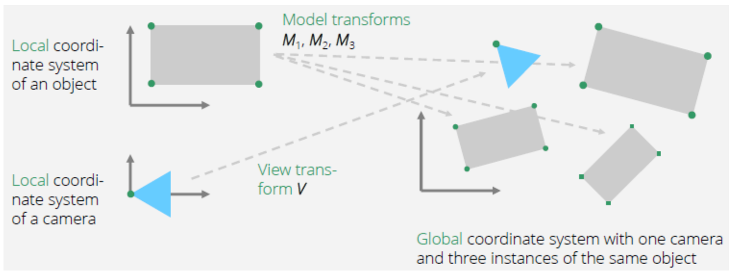
3.3 Shading Models

Illumination models can be evaluated per vertex or per fragment.
Faces/primitives are characterized by vertices.
Projected area of a face onto the sensor is subdivided into fragments.
Shading models specify whether the illumination model is evaluated per vertex or per fragment:
- If evaluated per vertex, the shading model specifies whether the resulting vertex colors are interpolated across a primitive or not.
- If evaluated per fragment, surface normals are interpolated across a primitive.

3.3.1 Well-knows Models

- Flat shading: evaluation per vertex, fragments are colored with the color of one specific vertex.
- Gouraud shading: evaluation per vertex, fragments are interpolated from vertex colors.
- Phong shading: evaluation per fragment, normals have to be interpolated from vertices to fragments.

4 Homogeneous Notation


Local coordinate system of an object
Model transforms M_1, M_2, M_3
Local coordinate system of a camera
View transform V
Global coordinate system with one camera and three instances of the same object

To transform from view space positions to positions on the camera plane:
- Projection transform
- Viewport transform
Affine trasformatiokns: angles and lengths are not preserved, collinearity, proportions, parallelism are.

3D position $\boldsymbol{p}$: $\boldsymbol{p}' = \boldsymbol{T}(\boldsymbol{p}) = \underbrace{\boldsymbol{A}}_{\text{Rotation scale ...}}\boldsymbol{p} + \underbrace{\boldsymbol{t}}_{\text{Translation}}$

3x3 matrix A represents linear transformations, 3D vector t represents translation. Using the homogeneous notation, all affine transformations are represented with one matrix vector multiplication.
Using the homogeneous notation, transformations of vectors and positions are handled in a unified way.
- From Cartesian to Homogeneous:

$(x, y, z)^\mathsf{T} \rightarrow [x, y, z, 1]^\mathsf{T}$ The most obvious way, but an infinite number of options.

$(x, y, z)^\mathsf{T} \rightarrow [\lambda x, \lambda y, \lambda z, \lambda]^\mathsf{T} \quad \lambda \neq 0$

- From Homogeneous to Cartesian:

$[x, y, z, w]^\mathsf{T} \rightarrow (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})^\mathsf{T}$

- General form:

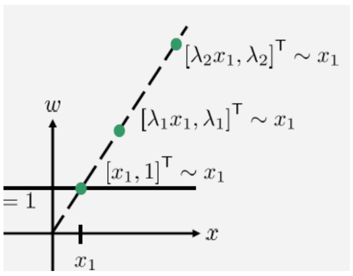

$[\lambda_2 x_1, \lambda_2]^\mathsf{T} \sim x_1$
w
$[\lambda_1 x_1, \lambda_1]^\mathsf{T} \sim x_1$
$[x_1, 1]^\mathsf{T} \sim x_1$
$= 1$
x
x_1

Figure 2: 1D Illustration

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & t_0 \\ m_{10} & m_{11} & m_{12} & t_1 \\ m_{20} & m_{21} & m_{22} & t_2 \\ p_0 & p_1 & p_2 & w \end{bmatrix}$$

m is rotation,scale,shear; t is translation; p for projection; w is the homogeneous component.

4.1 Transformations

4.1.1 Translation

– Of a position

$$\boldsymbol{T}(\boldsymbol{t})\boldsymbol{p} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix}$$

– Of a vector

$$\boldsymbol{T}(\boldsymbol{t})\boldsymbol{v} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

– Inverse transform

$$\boldsymbol{T}^{-1}(\boldsymbol{t}) = \boldsymbol{T}(-\boldsymbol{t})$$

4.1.2 Rotation

Positive (anticlockwise)

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R}_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R}_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.1.3 Mirroring

$$\boldsymbol{P}_x = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{P}_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{P}_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation and reflection matrices are orthogonal: $RR^T = R^T R = I, R^-1 = R^T$

4.1.4 Scale

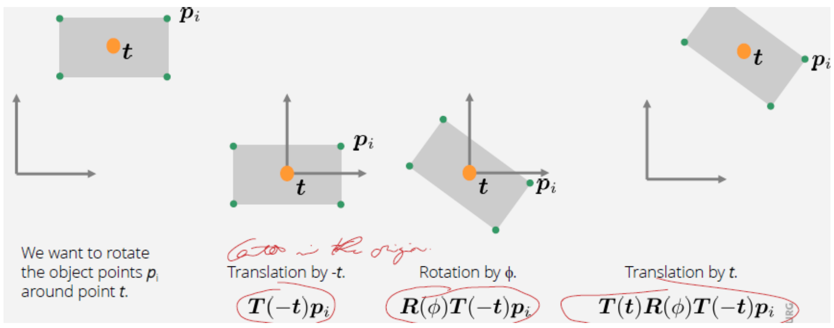
$$\boldsymbol{S}(s_x, s_y, s_z)\boldsymbol{p} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \\ 1 \end{bmatrix}$$

4.1.5 Shear

Offset of one component wrt another component.

$$\boldsymbol{H}_{xz}(s)\boldsymbol{p} = \begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + s p_z \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

Example:

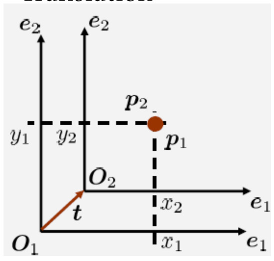

$\boldsymbol{p}_i$
$\boldsymbol{t}$
We want to rotate the object points p_i around point t.
Go to the origin.
Translation by $-t$.
$\boldsymbol{T}(-\boldsymbol{t})\boldsymbol{p}_i$
Rotation by ϕ.
$\boldsymbol{R}(\phi)\boldsymbol{T}(-\boldsymbol{t})\boldsymbol{p}_i$
Translation by t.
$\boldsymbol{T}(\boldsymbol{t})\boldsymbol{R}(\phi)\boldsymbol{T}(-\boldsymbol{t})\boldsymbol{p}_i$

4.1.6 Planes and normals

Planes can be represented by a surface normal n and a point r. All points p with n*(p-r)=0 form a plane.

4.1.7 Basis Transform

- Translation



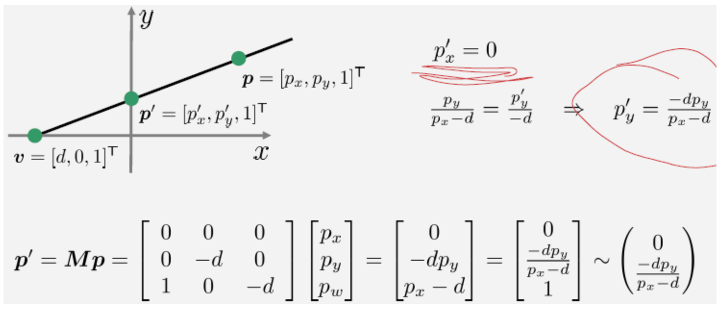

y
x
$\boldsymbol{p} = [p_x, p_y, 1]^\top$
$\boldsymbol{p}' = [p'_x, p'_y, 1]^\top$
$\boldsymbol{v} = [d, 0, 1]^\top$
$p'_x = 0$
$\frac{p_y}{p_x - d} = \frac{p'_y}{-d} \Rightarrow p'_y = \frac{-dp_y}{p_x - d}$
$\boldsymbol{p}' = \boldsymbol{M}\boldsymbol{p} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -d & 0 \\ 1 & 0 & -d \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_w \end{bmatrix} = \begin{bmatrix} 0 \\ -dp_y \\ p_x - d \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{-dp_y}{p_x - d} \\ 1 \end{bmatrix} \sim \begin{pmatrix} 0 \\ \frac{-dp_y}{p_x - d} \end{pmatrix}$

Figure 3: To compute intersection with the x-axis

- Rotation

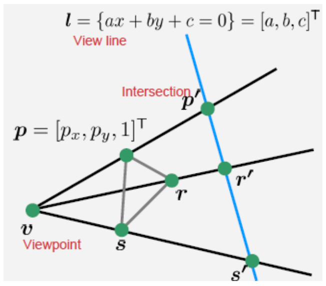
$$\boldsymbol{p}_2 = \begin{pmatrix} \boldsymbol{b}_1^\top \\ \boldsymbol{b}_2^\top \\ \boldsymbol{b}_3^\top \end{pmatrix} \boldsymbol{p}_1 \sim \begin{bmatrix} \boldsymbol{b}_{1,x} & \boldsymbol{b}_{1,y} & \boldsymbol{b}_{1,z} & 0 \\ \boldsymbol{b}_{2,x} & \boldsymbol{b}_{2,y} & \boldsymbol{b}_{2,z} & 0 \\ \boldsymbol{b}_{3,x} & \boldsymbol{b}_{3,y} & \boldsymbol{b}_{3,z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{p}_1$$

The view transform can be seen as a basis transform. Placing and orienting the camera is a transformation v.The basis transform is realized by applying v^-1 to all objects.
Usage of the homogeneous notation is motivated by a unified processing of affine transformations, perspective projections, points, and vectors.All transformations of points and vectors are represented by a matrix vector multiplication.

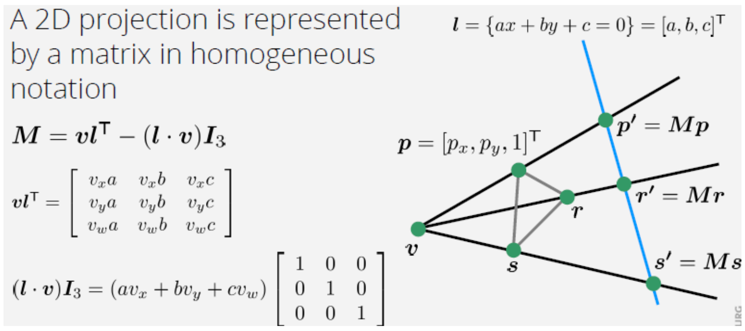
5 Projection

Last matrix M row is dedicated to projections and can be used to realize divisions by a linear combination.

5.1 2D projections



If the homogeneous component of v is not equal to zero, we have a perspective projection; if v is at infinity, we have a parallel projection.
Location of viewpoint and orientation of the viewline determine the type of projection:
- Parallel (viewpoint at infinity, parallel projectors).
- Orthographic (viewline orthogonal to the projectors).
- Oblique (viewline not orthogonal to the projectors).
- Perspective (non parallel projectors):
- One point (viewline intersects one principal axis).
- Two point (viewline intersects two principal axes, two vanishing points).



A 2D projection is represented by a matrix in homogeneous notation

$$\boldsymbol{M} = \boldsymbol{v}\boldsymbol{l}^{\mathsf{T}} - (\boldsymbol{l}\cdot\boldsymbol{v})\boldsymbol{I}_3$$

$$\boldsymbol{v}\boldsymbol{l}^{\mathsf{T}} = \begin{bmatrix} v_x a & v_x b & v_x c \\ v_y a & v_y b & v_y c \\ v_w a & v_w b & v_w c \end{bmatrix}$$

$$(\boldsymbol{l}\cdot\boldsymbol{v})\boldsymbol{I}_3 = (a v_x + b v_y + c v_w)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

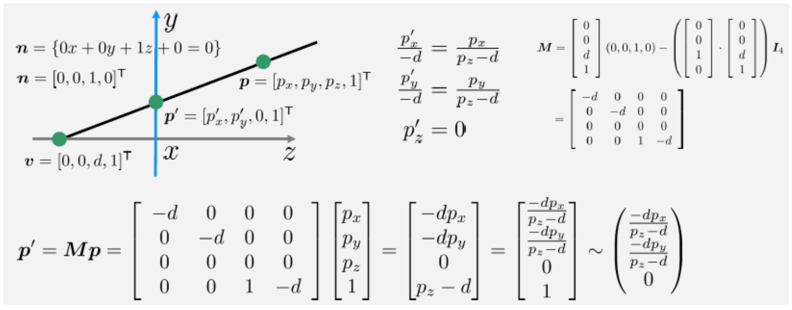
$$\boldsymbol{l} = \{ax + by + c = 0\} = [a, b, c]^{\mathsf{T}}$$

$\boldsymbol{p} = [p_x, p_y, 1]^{\mathsf{T}}$
$\boldsymbol{p}' = \boldsymbol{M}\boldsymbol{p}$
$\boldsymbol{r}' = \boldsymbol{M}\boldsymbol{r}$
$\boldsymbol{r}$
$\boldsymbol{v}$
$\boldsymbol{s}$
$\boldsymbol{s}' = \boldsymbol{M}\boldsymbol{s}$

See other examples on slides.

5.2 3D projections

Same thing expanded to 3D.



$\boldsymbol{n} = \{0x + 0y + 1z + 0 = 0\}$
$\boldsymbol{n} = [0, 0, 1, 0]^{\mathsf{T}}$
y
$\boldsymbol{p} = [p_x, p_y, p_z, 1]^{\mathsf{T}}$
$\boldsymbol{p}' = [p'_x, p'_y, 0, 1]^{\mathsf{T}}$
$\boldsymbol{v} = [0, 0, d, 1]^{\mathsf{T}}$
x
z

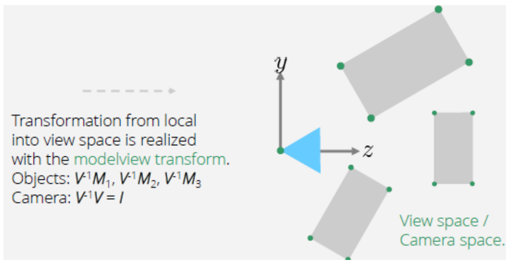
$$\frac{p'_x}{-d} = \frac{p_x}{p_z - d} \qquad \frac{p'_y}{-d} = \frac{p_y}{p_z - d} \qquad p'_z = 0$$

$$\boldsymbol{M} = \begin{bmatrix} 0 \\ 0 \\ d \\ 1 \end{bmatrix}(0,0,1,0) - \left(\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}\cdot\begin{bmatrix} 0 \\ 0 \\ d \\ 1 \end{bmatrix}\right)\boldsymbol{I}_4 = \begin{bmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -d \end{bmatrix}$$

$$\boldsymbol{p}' = \boldsymbol{M}\boldsymbol{p} = \begin{bmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -d \end{bmatrix}\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} -dp_x \\ -dp_y \\ 0 \\ p_z - d \end{bmatrix} = \begin{bmatrix} \frac{-dp_x}{p_z - d} \\ \frac{-dp_y}{p_z - d} \\ 0 \\ 1 \end{bmatrix} \sim \begin{pmatrix} \frac{-dp_x}{p_z - d} \\ \frac{-dp_y}{p_z - d} \\ 0 \end{pmatrix}$$

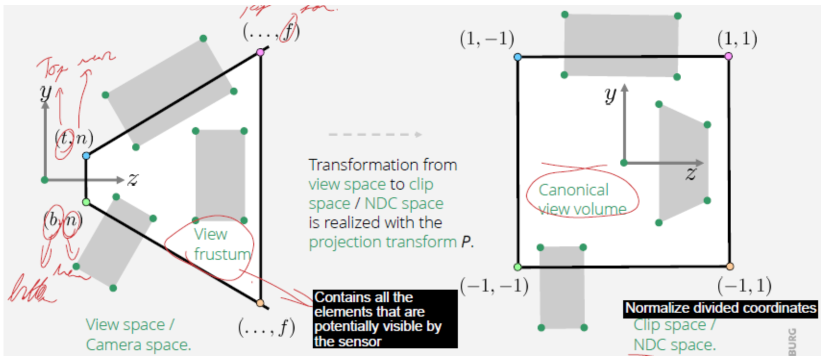
6 Projection Transform

6.1 Modelview Transformation



Transformation from local into view space is realized with the modelview transform.
Objects: $V^{-1}M_1$, $V^{-1}M_2$, $V^{-1}M_3$
Camera: $V^{-1}V = I$

6.2 Projection Transformation



Transformation from view space to clip space / NDC space is realized with the projection transform *P*.

Contains all the elements that are potentially visible by the sensor

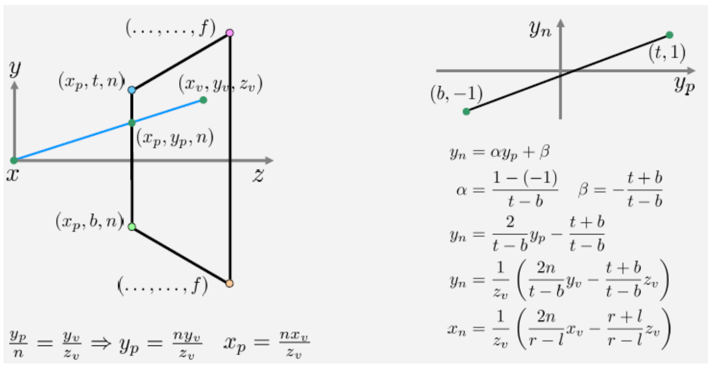
Normalize divided coordinates

Why to transform to Clip/NDC? It allows simplified and unified implementations:
Culling (exclude non-visible elements), Clipping (exclude non-visible parts of visible elements), Visibility (useful for ray tracing).

6.3 Perspective Projection Transformation

Maps a view volume/pyramidal frustum (l,r,t,b) to a canonical view volume (-1,1). It is applied to vertices. Maps coordinates to (-1,1) maintaining coherence.

- Derivation



$$\frac{y_p}{n} = \frac{y_v}{z_v} \Rightarrow y_p = \frac{n y_v}{z_v} \qquad x_p = \frac{n x_v}{z_v}$$

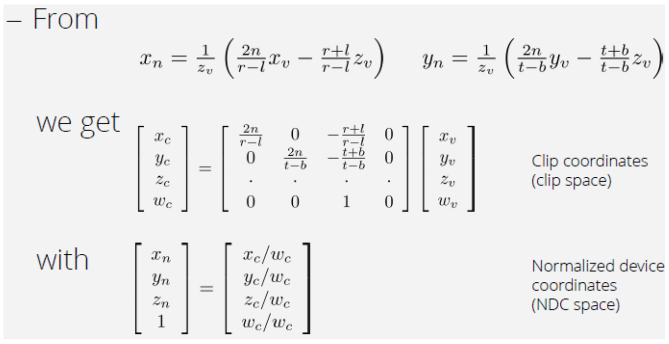
$$y_n = \alpha y_p + \beta$$

$$\alpha = \frac{1-(-1)}{t-b} \qquad \beta = -\frac{t+b}{t-b}$$

$$y_n = \frac{2}{t-b} y_p - \frac{t+b}{t-b}$$

$$y_n = \frac{1}{z_v}\left(\frac{2n}{t-b} y_v - \frac{t+b}{t-b} z_v\right)$$

$$x_n = \frac{1}{z_v}\left(\frac{2n}{r-l} x_v - \frac{r+l}{r-l} z_v\right)$$



– From

$$x_n = \frac{1}{z_v}\left(\frac{2n}{r-l} x_v - \frac{r+l}{r-l} z_v\right) \qquad y_n = \frac{1}{z_v}\left(\frac{2n}{t-b} y_v - \frac{t+b}{t-b} z_v\right)$$

we get

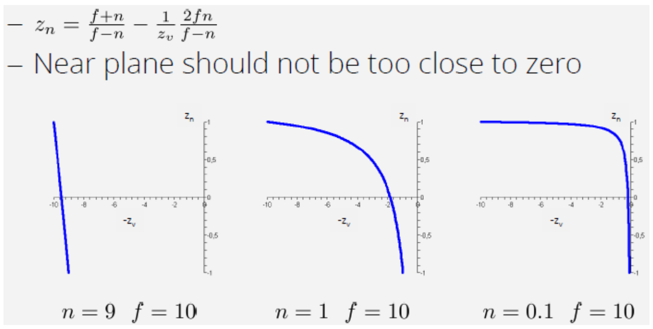
$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_v \\ y_v \\ z_v \\ w_v \end{bmatrix}$$

Clip coordinates (clip space)

with

$$\begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix} = \begin{bmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \\ w_c/w_c \end{bmatrix}$$

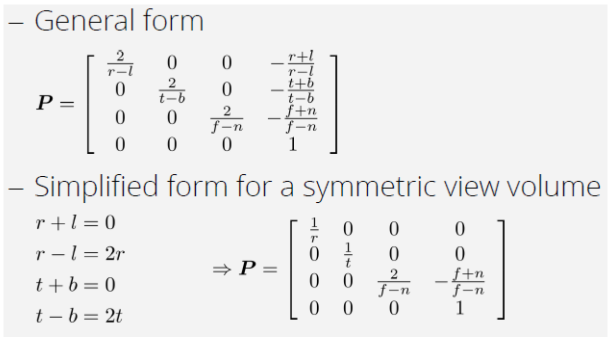
Normalized device coordinates (NDC space)



– $z_n = \frac{f+n}{f-n} - \frac{1}{z_v}\frac{2fn}{f-n}$

– Near plane should not be too close to zero

Figure 4: Non-linear mapping of depth values



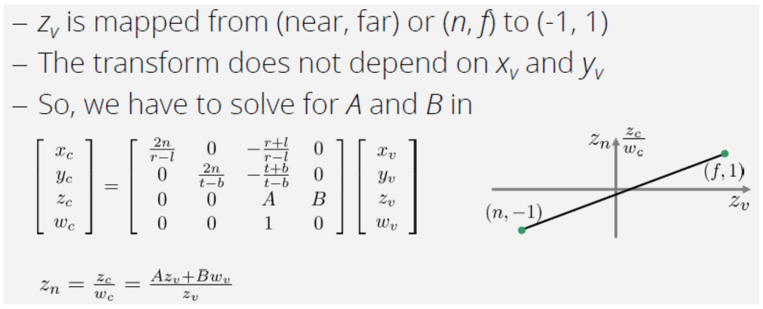
– General form

$$\boldsymbol{P} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– Simplified form for a symmetric view volume

$$\begin{aligned} r+l &= 0 \\ r-l &= 2r \\ t+b &= 0 \\ t-b &= 2t \end{aligned} \qquad \Rightarrow \boldsymbol{P} = \begin{bmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 5: Matrix general form



– z_v is mapped from (near, far) or (n, f) to (-1, 1)

– The transform does not depend on x_v and y_v

– So, we have to solve for A and B in

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_v \\ y_v \\ z_v \\ w_v \end{bmatrix}$$

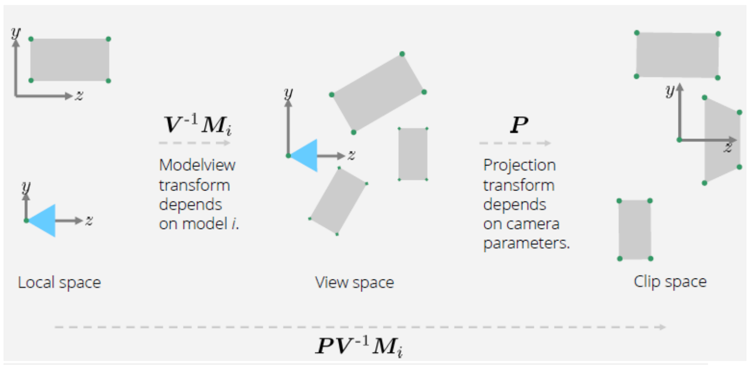
$$z_n = \frac{z_c}{w_c} = \frac{Az_v + Bw_v}{z_v}$$

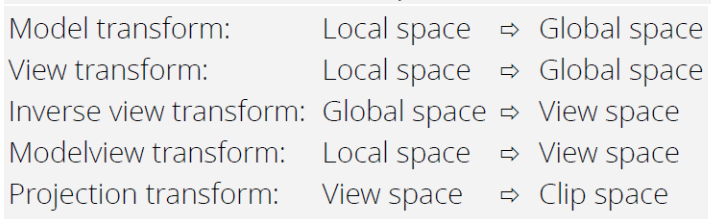
$z_n \frac{z_c}{w_c}$
$(f, 1)$
z_v
$(n, -1)$

Then, we can solve A and B and get the complete Projection Matrix

6.4 Orthographic Projection

7 Typical vertex transformations


y
z
$V^{-1}M_i$
Modelview transform depends on model i.
P
Projection transform depends on camera parameters.
Local space
View space
Clip space
$PV^{-1}M_i$



Model transform:	Local space	⇨	Global space
View transform:	Local space	⇨	Global space
Inverse view transform:	Global space	⇨	View space
Modelview transform:	Local space	⇨	View space
Projection transform:	View space	⇨	Clip space

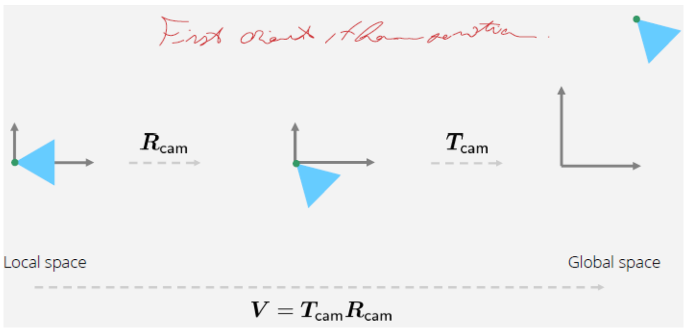

First orient, then position.
R_{cam}
T_{cam}
Local space
Global space
$V = T_{\text{cam}} R_{\text{cam}}$

Figure 6: Camera placement

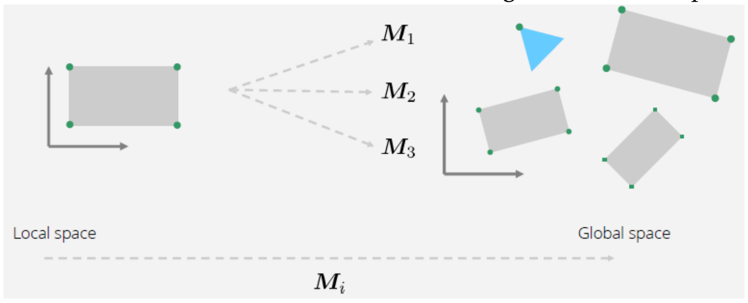


Figure 7: Object Placement

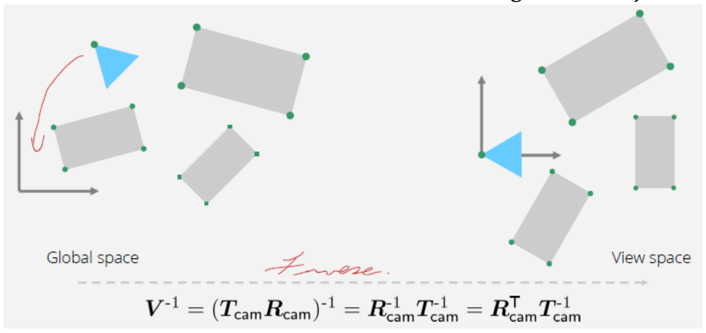
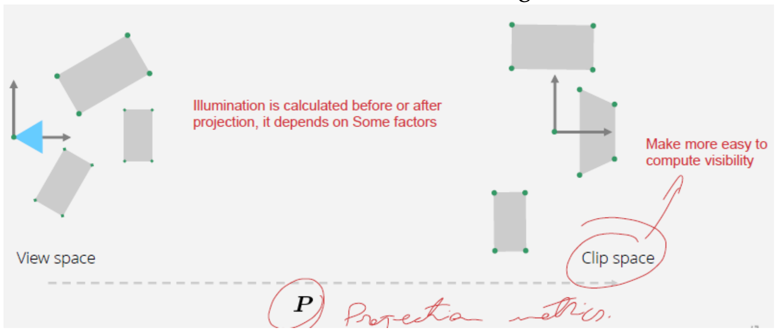

Global space
Inverse.
View space
$V^{-1} = (T_{\text{cam}} R_{\text{cam}})^{-1} = R_{\text{cam}}^{-1} T_{\text{cam}}^{-1} = R_{\text{cam}}^{\mathsf{T}} T_{\text{cam}}^{-1}$

Figure 8: View Transform


Illumination is calculated before or after projection, it depends on Some factors

Make more easy to compute visibility

Figure 9: Projection transform