INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 3: “Ray Tracing (Introduction)”

Welcome!

Today’s Agenda:

- Primitives *(contd.)*
- Ray Tracing
- Intersections
- Assignment 2
- Textures

Previously in INFOGR

Primitives

Implicit curves: $f(x, y) = 0$

Circle: $x^2 + y^2 - r^2 = 0$
Line: $Ax + By + C = 0$

Slope-intersect form of a line: $y = ax + c$

Normal of line $Ax + By + C = 0$: $\vec{N} = \begin{pmatrix} A \\ B \end{pmatrix}$

Distance of line $Ax + By + C = 0$ to the origin: $|C|$ (if $\left\|\vec{N}\right\| = 1$).

Primitives

Parametric representation

Parametric curve:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} g(t) \\ h(t) \end{pmatrix}$$

Example: line

$$p_0 = (x_{p_0}, y_{p_0}), p_1 = (x_{p_1}, y_{p_1})$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_{p_0} \\ y_{p_0} \end{pmatrix} + t \begin{pmatrix} x_{p_1} - x_{p_0} \\ y_{p_1} - y_{p_0} \end{pmatrix}$$

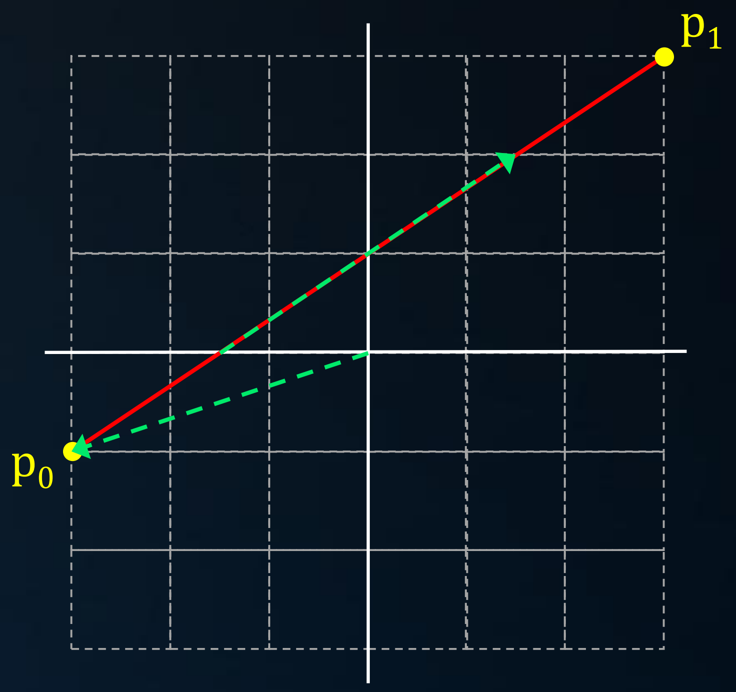
Or

$$p(t) = p_0 + t(p_1 - p_0), t \in \mathbb{R}.$$

In this example:

p_0 is the *support vector;*

$p_1 - p_0$ is the *direction vector.*



Primitives

Slope-intercept:

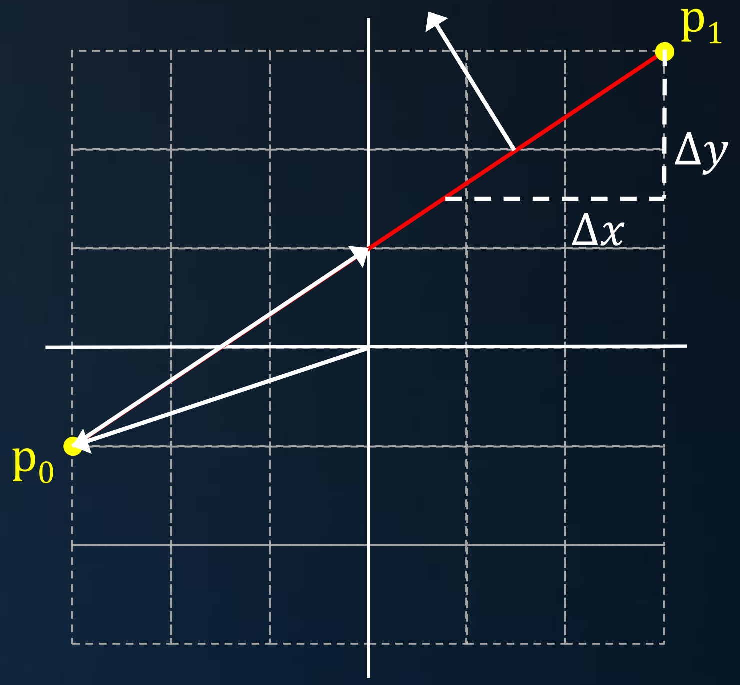
$$y = ax + c$$

Implicit representation:

$$-ax + y - c = 0$$
$$Ax + By + C = 0$$

Parametric representation:

$$p(t) = p_0 + t(p_1 - p_0)$$

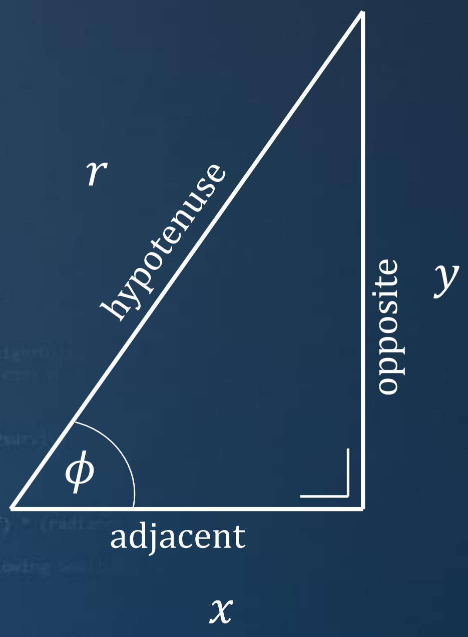


Primitives

Circle - parametric

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_c + r\cos\phi \\ y_c + r\sin\phi \end{pmatrix}$$

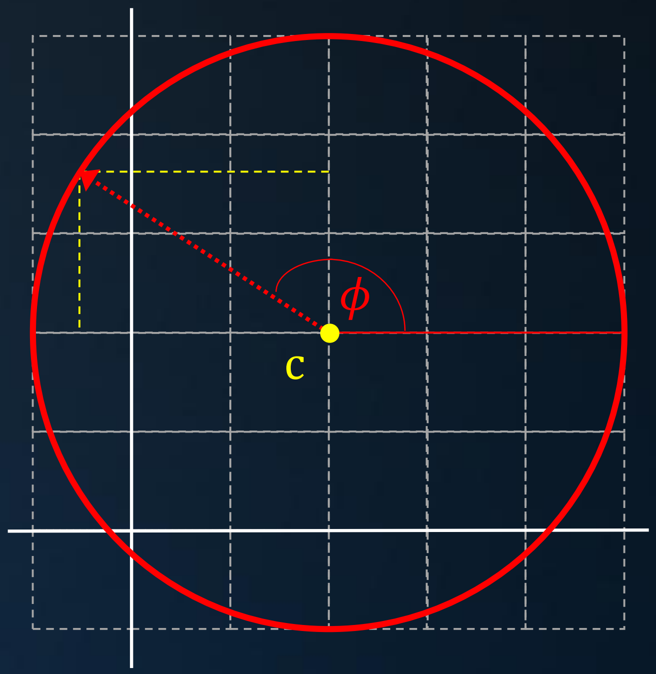
ϕ = “phi”



$$\cos\phi = \frac{x}{r}$$

$$sin\,\phi = \frac{y}{r}$$

$$tan\,\phi = \frac{y}{x}$$



SOH CAH TOA

Primitives

Circle – sphere (implicit)

Recall: the implicit representation for a circle with radius r and center c is:

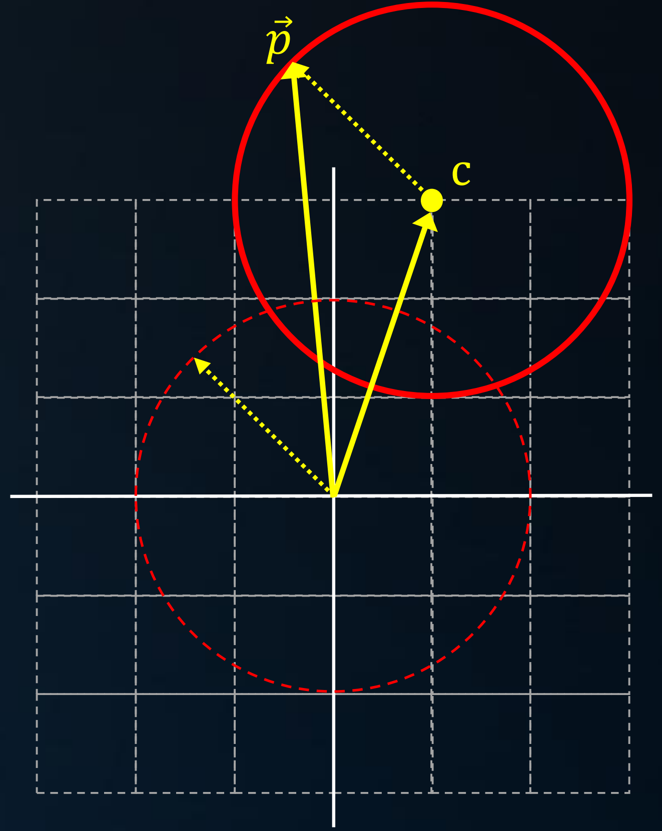
$$(x - c_x)^2 + (y - c_y)^2 - r^2 = 0$$

or: $\| \mathrm{p} - \mathrm{c} \|^2 - r^2 = 0$ ➔ $\| p - c \| = r$

In $\mathbb{R}^3$, we get:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - r^2 = 0$$

or: $\| p - c \| = r$



Primitives

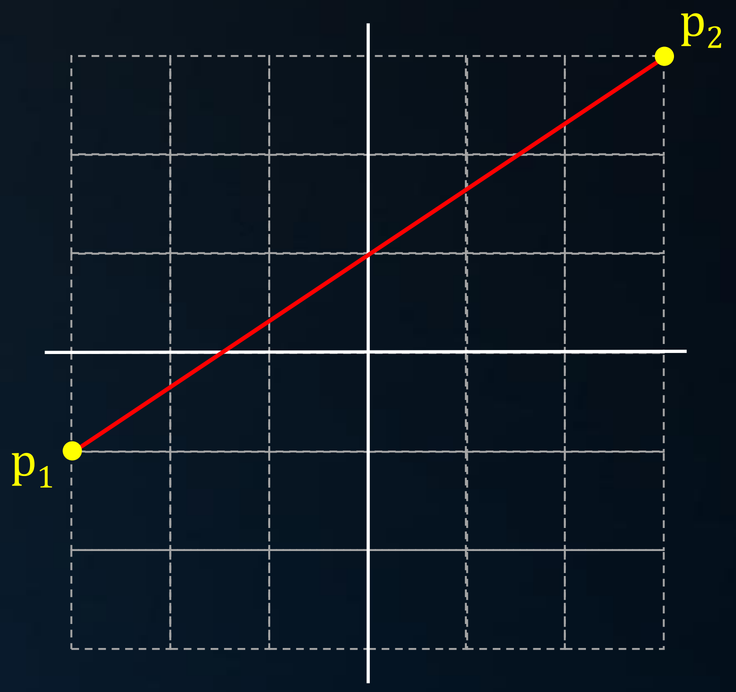
Line – plane (implicit)

Recall: the implicit representation for a line is:

$$Ax + By + C = 0$$

In $\mathbb{R}^3$, we get a plane:

$$Ax + By + Cz + D = 0$$



Primitives

Parametric surfaces

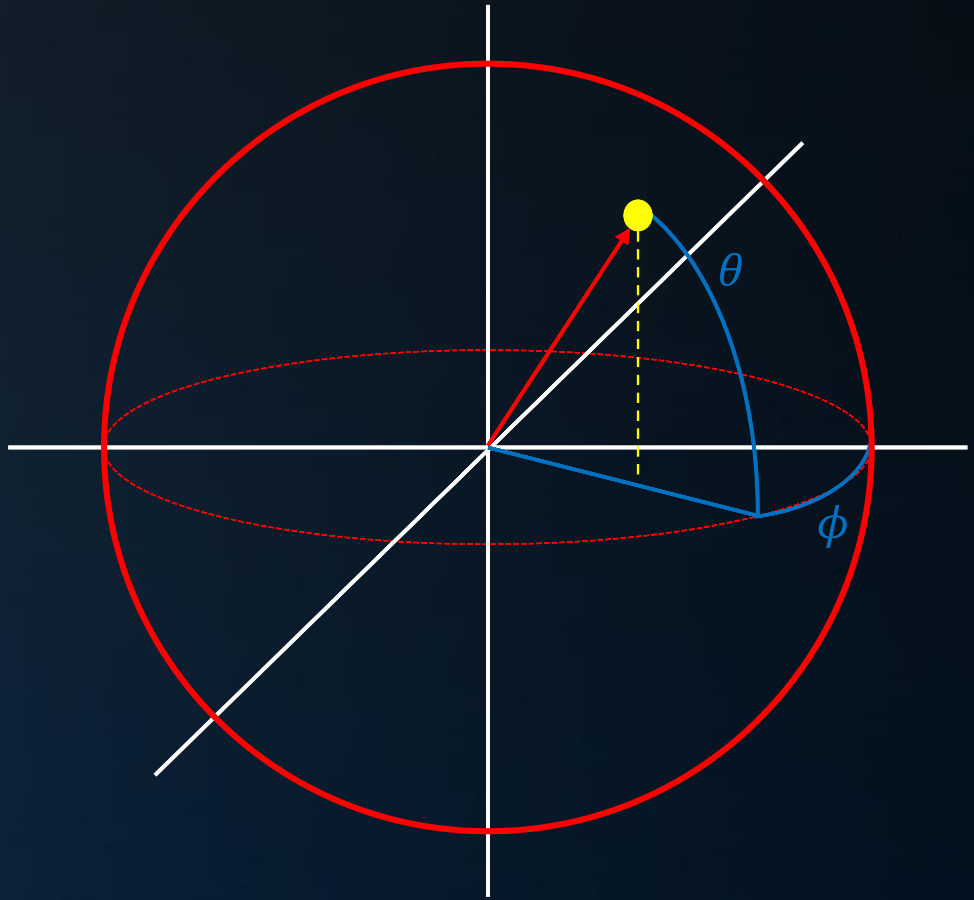
A parametric surface in $\mathbb{R}^3$ needs two parameters:

$$x = f(u, v),$$
$$y = g(u, v),$$
$$z = h(u, v).$$

For example, a sphere:

$$x = r\cos\phi\sin\theta,$$
$$y = r\sin\phi\sin\theta,$$
$$z = r\cos\theta.$$

Doesn’t look very convenient (compared to the implicit form), but it will prove useful for texture mapping.



Primitives

Parametric planes

Recall the parametric line definition:

$$p(t) = p_0 + t(p_1 - p_0)$$

For a plane, we need to parameters:

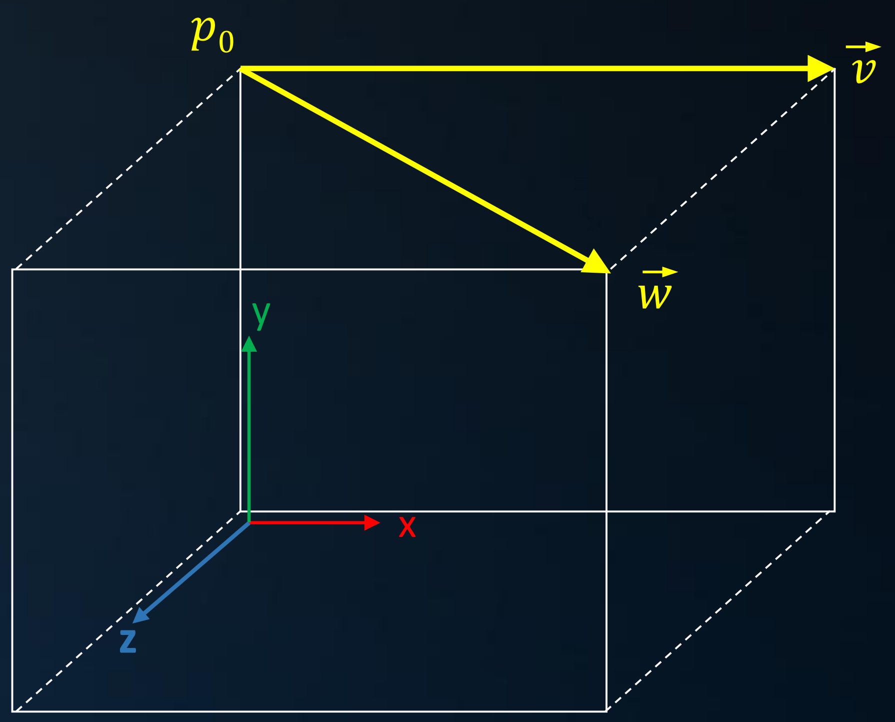
$$p(s,t) = p_0 + s(p_1 - p_0) + t(p_2 - p_0)$$

or:

$$p(s,t) = p_0 + s\vec{v} + t\vec{w}$$

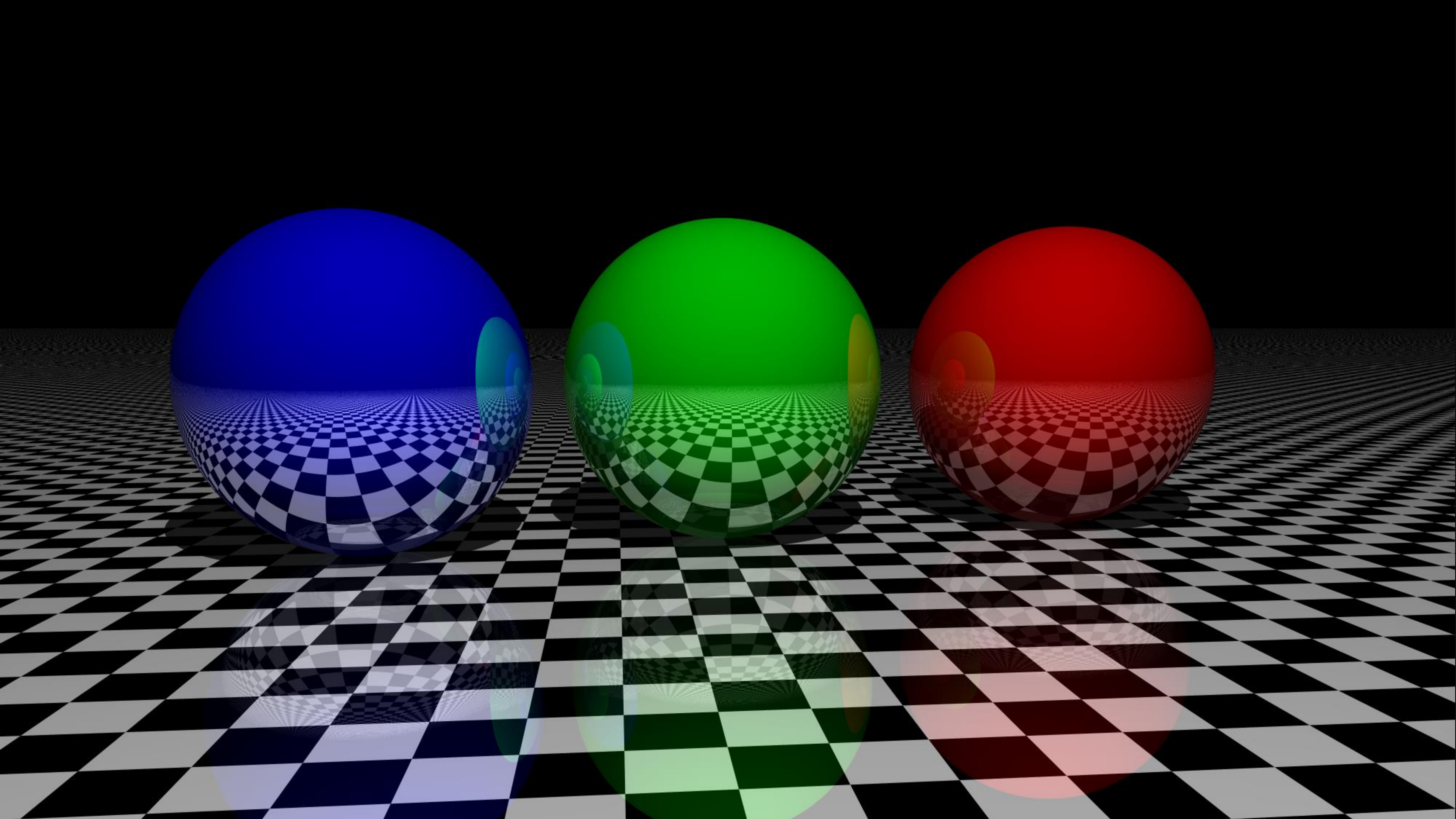
where:

- p_0 is a point on the plane;
- $\vec{v}$ and $\vec{w}$ are two linearly independent vectors on the plane;
- $s, t \in \mathbb{R}$.



Today's Agenda:

- Primitives *(contd.)*
- Ray Tracing
- Intersections
- Assignment 2
- Textures



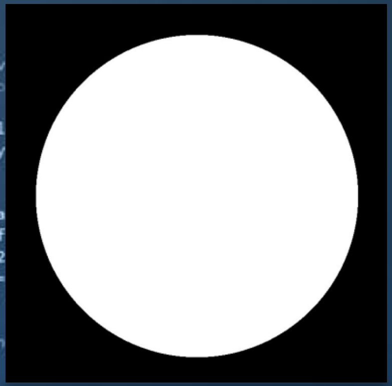
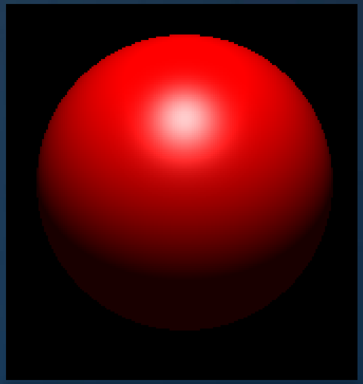
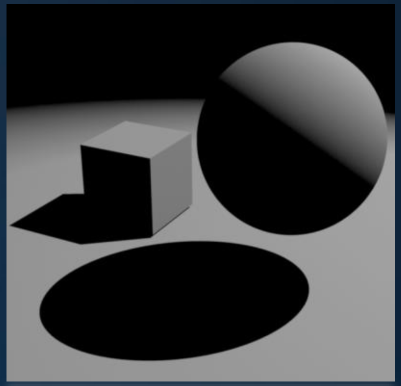
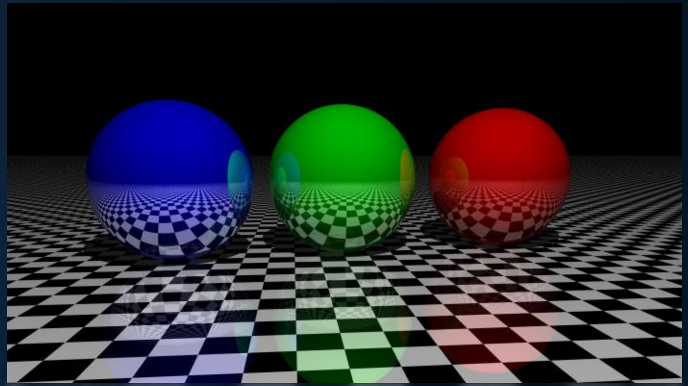
Ray Tracing

PART 1: Introduction (today)

PART 2: Shading (May 10)

PART 3: Reflections, refraction, absorption (May 17)

PART 4: Path Tracing (June 21)

Ray Tracing

Ray Tracing:

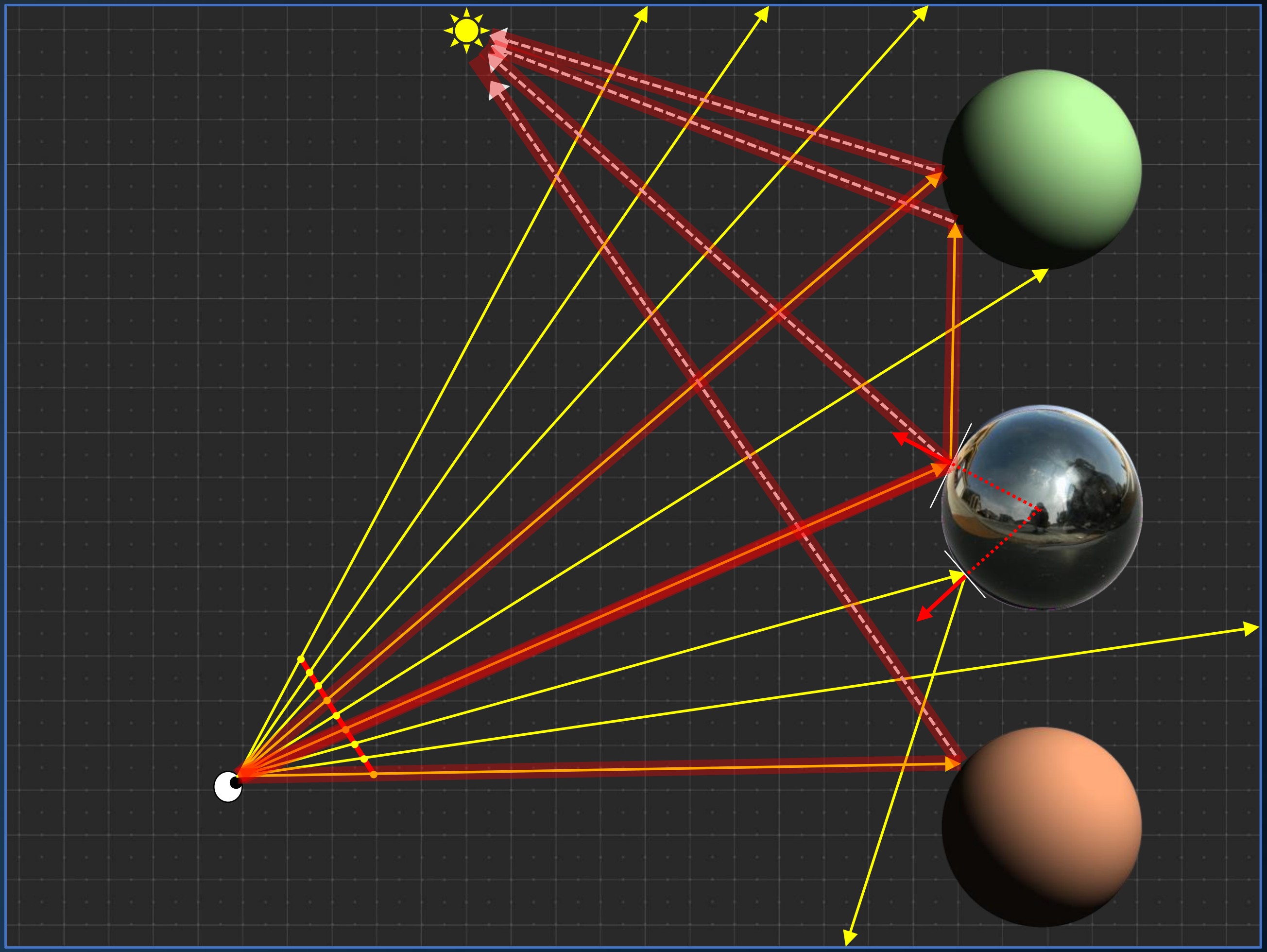
World space

- Geometry
- Eye
- Screen plane
- Screen pixels
- Primary rays
- Intersections
- Point light
- Shadow rays

Light transport

- Extension rays

Light transport



Ray Tracing

Ray Tracing:

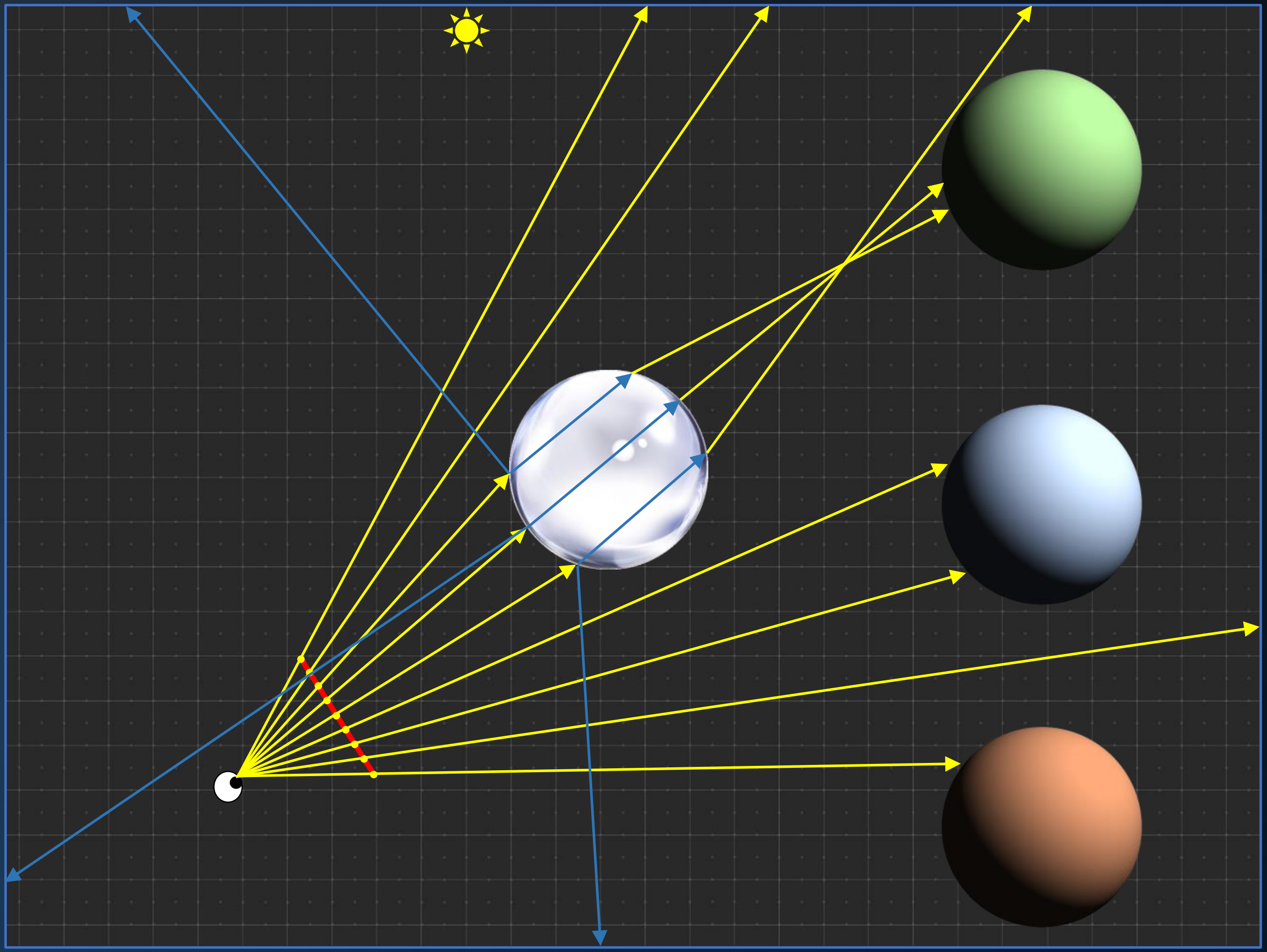
World space

- Geometry
- Eye
- Screen plane
- Screen pixels
- Primary rays
- Intersections
- Point light
- Shadow rays

Light transport

- Extension rays

Light transport



Ray Tracing

Ray Tracing:

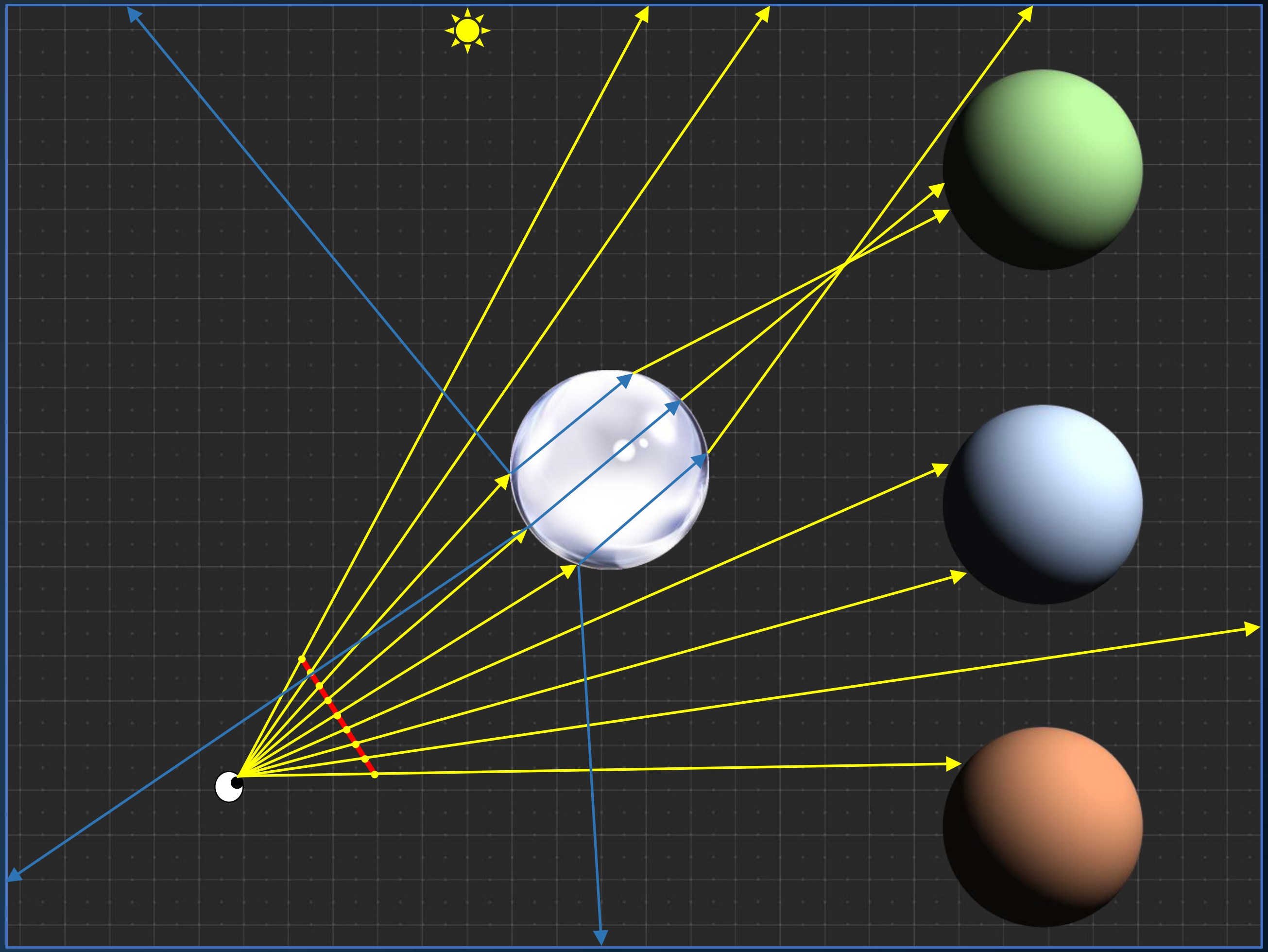
World space

- Geometry
- Eye
- Screen plane
- Screen pixels
- Primary rays
- Intersections
- Point light
- Shadow rays

Light transport

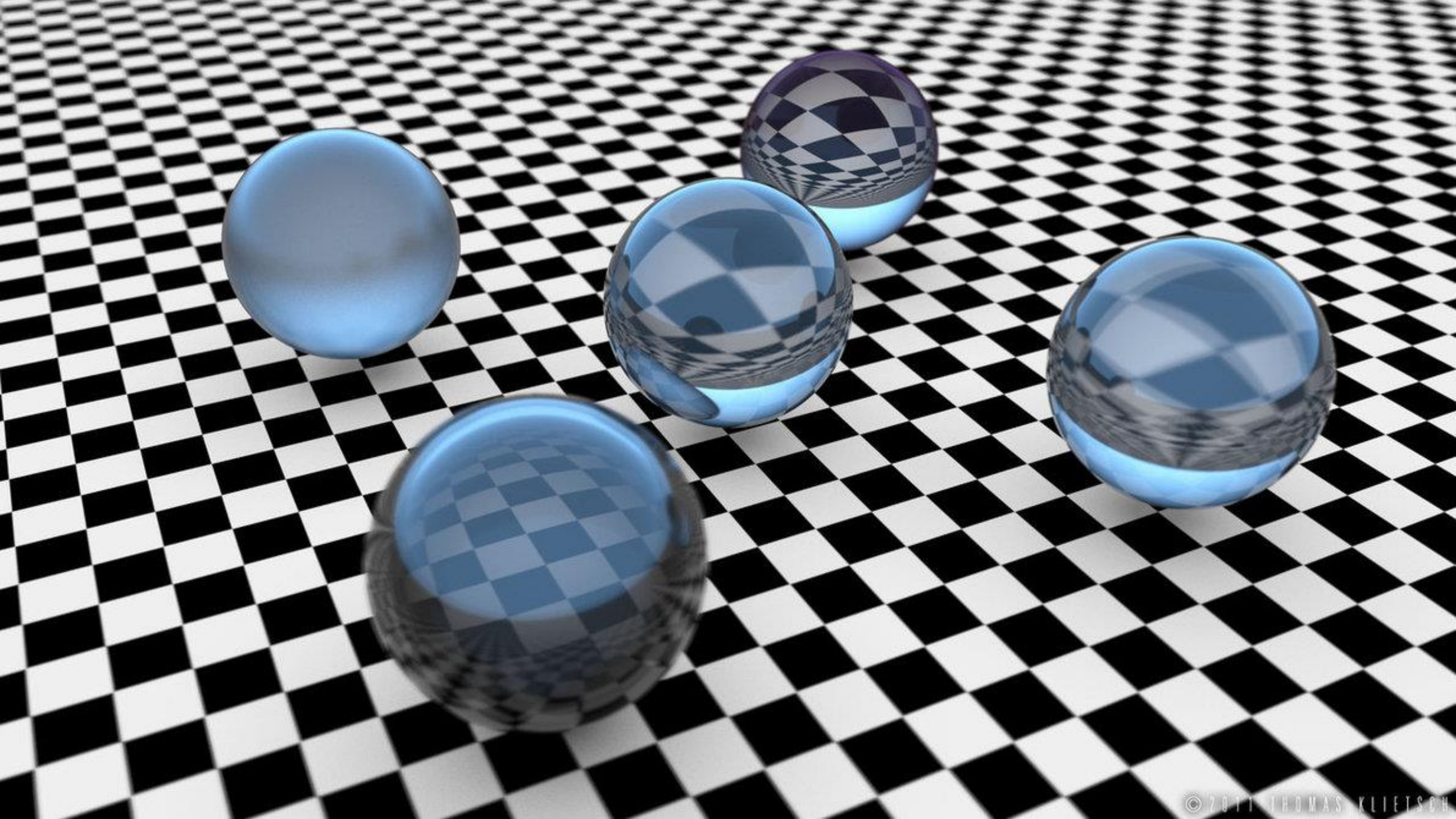
- Extension rays

Light transport

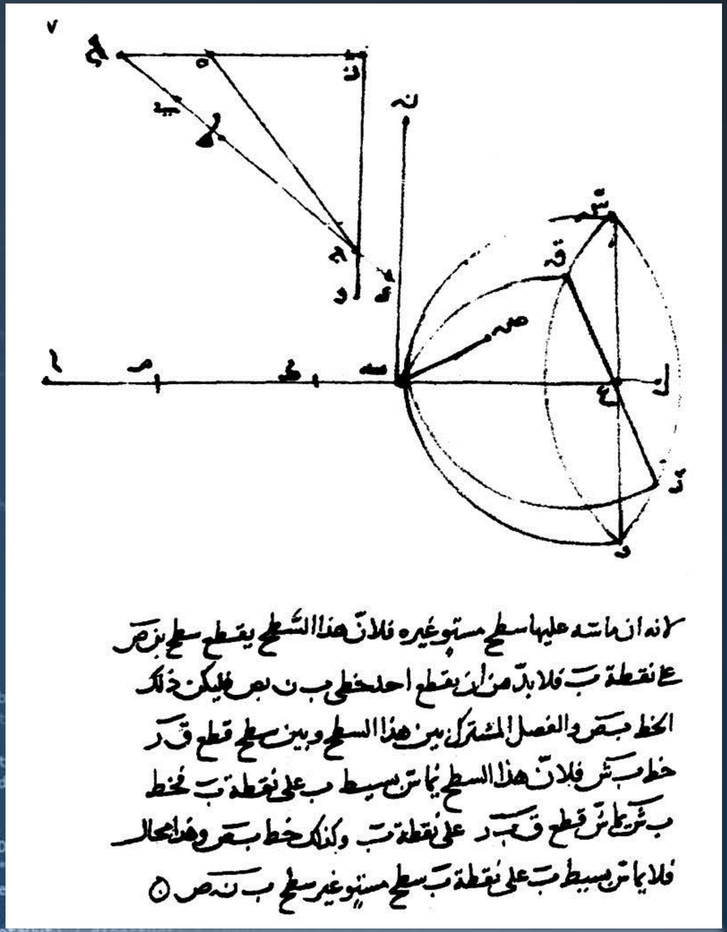


Note:

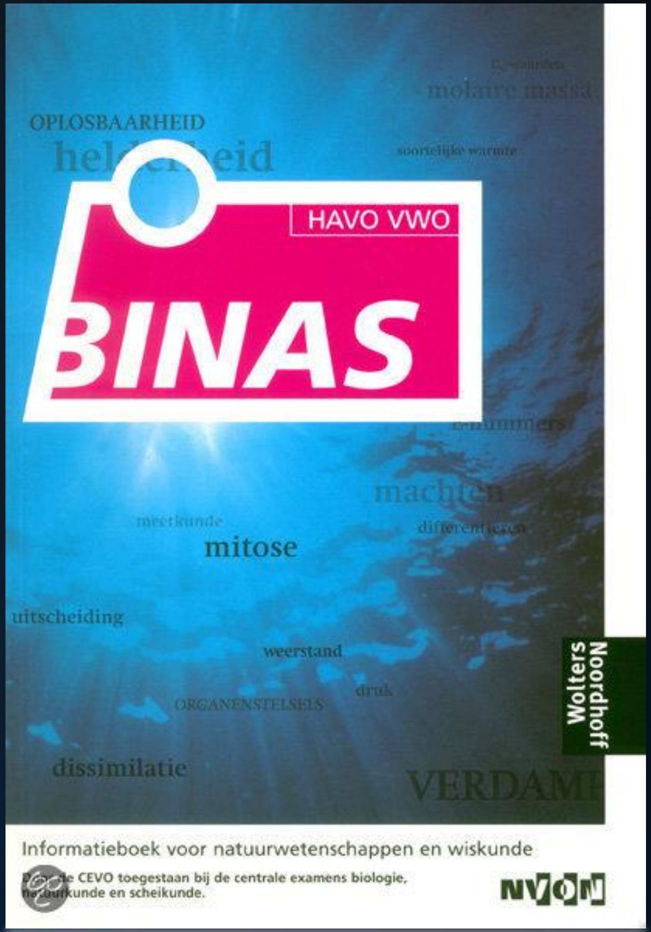
We are calculating light transport *backwards.*


©2011 THOMAS KLIETSCH

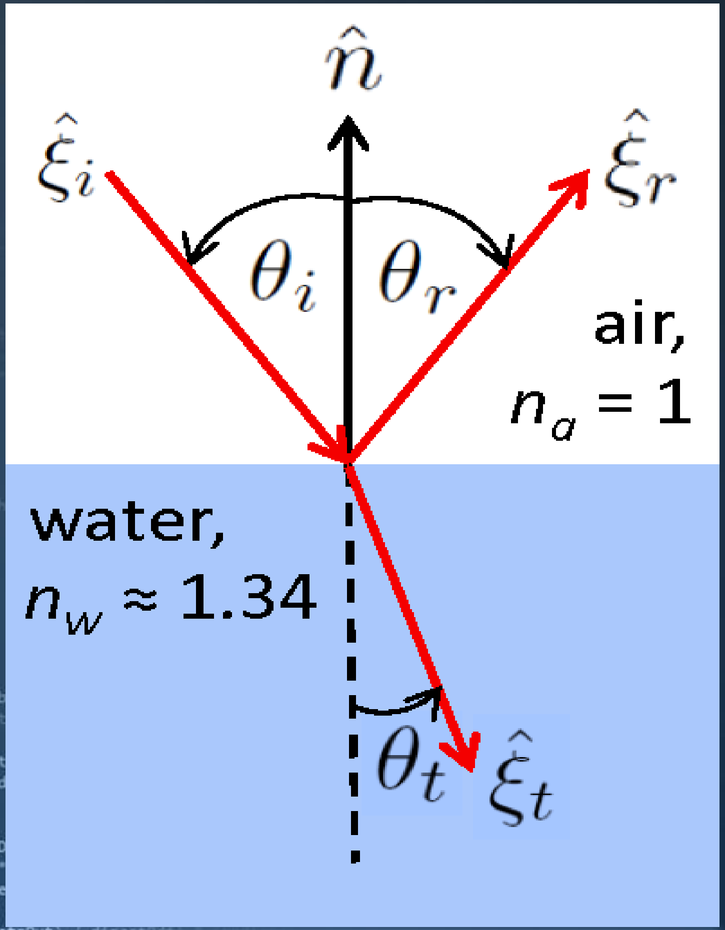
Ray Tracing

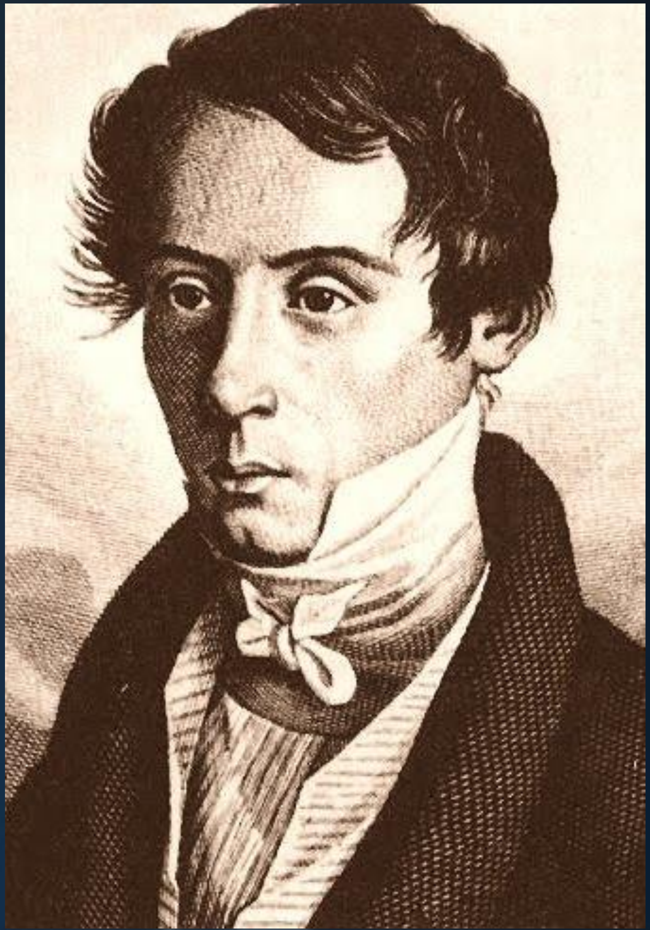


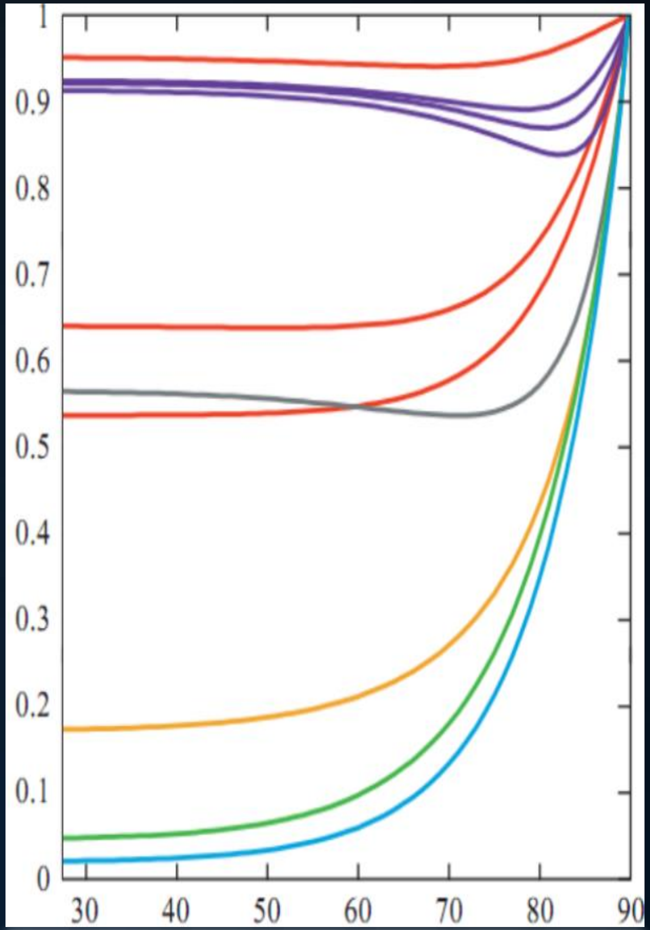

WILLEBRORDUS SNELLIUS.
Matheseos Professor.


OPLOSBAARHEID
HAVO VWO
BINAS
mitose
uitscheiding
weerstand
dissimilatie
Wolters Noordhoff
Informatieboek voor natuurwetenschappen en wiskunde
NVON

Ray Tracing


$\hat{n}$
$\hat{\xi}_i$
$\hat{\xi}_r$
θ_i
θ_r
air,
$n_a = 1$
water,
$n_w \approx 1.34$
θ_t
$\hat{\xi}_t$




1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
30
40
50
60
70
80
90

Ray Tracing

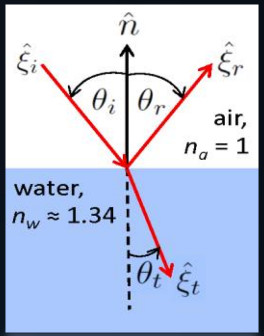
Physical basis

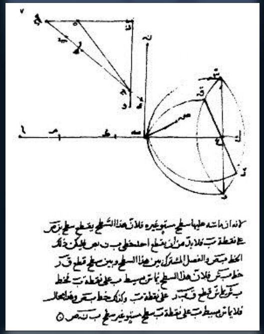
Ray tracing uses *ray optics* to simulate the behavior of light in a virtual environment.

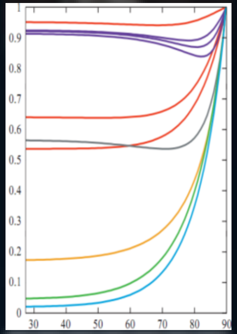
It does so by finding light transport paths:

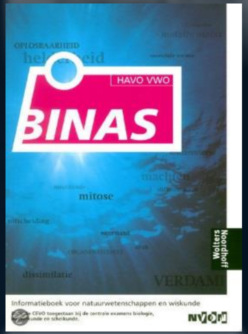
- From the 'eye'
- Through a pixel
- Via scene surfaces
- To one or more light sources.

At each surface, the light is modulated. The final value is deposited at the pixel (simulating reception by a sensor).









Today's Agenda:

- Primitives *(contd.)*
- Ray Tracing
- Intersections
- Assignment 2
- Textures

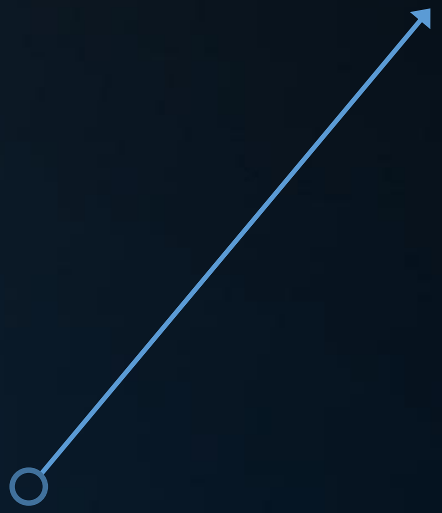
Intersections

Ray definition

A ray is an infinite line with a start point:

$p(t) = O + t\vec{D}$, where $t > 0$.

```
struct Ray
{
    float3 O;    // ray origin
    float3 D;    // ray direction
    float t;     // distance
};
```



The ray direction is generally *normalized*.

Intersect

Ray setup

A ray is initially shot through a pixel on the screen plane.
The screen plane is defined in world space:

Camera position: $E = (0,0,0)$
View direction: $\vec{V}$
Screen center: $C = E + d\vec{V}$
Screen corners: $p_0 = C + (-1,-1,0)$, $p_1 = C + (1,-1,0)$, $p_2 = C + (-1,1,0)$

From here:

- Change FOV by altering d;
- Transform camera by multiplying E, p_0, p_1, p_2 with the camera matrix.

Intersect

Ray setup

Point on the screen:

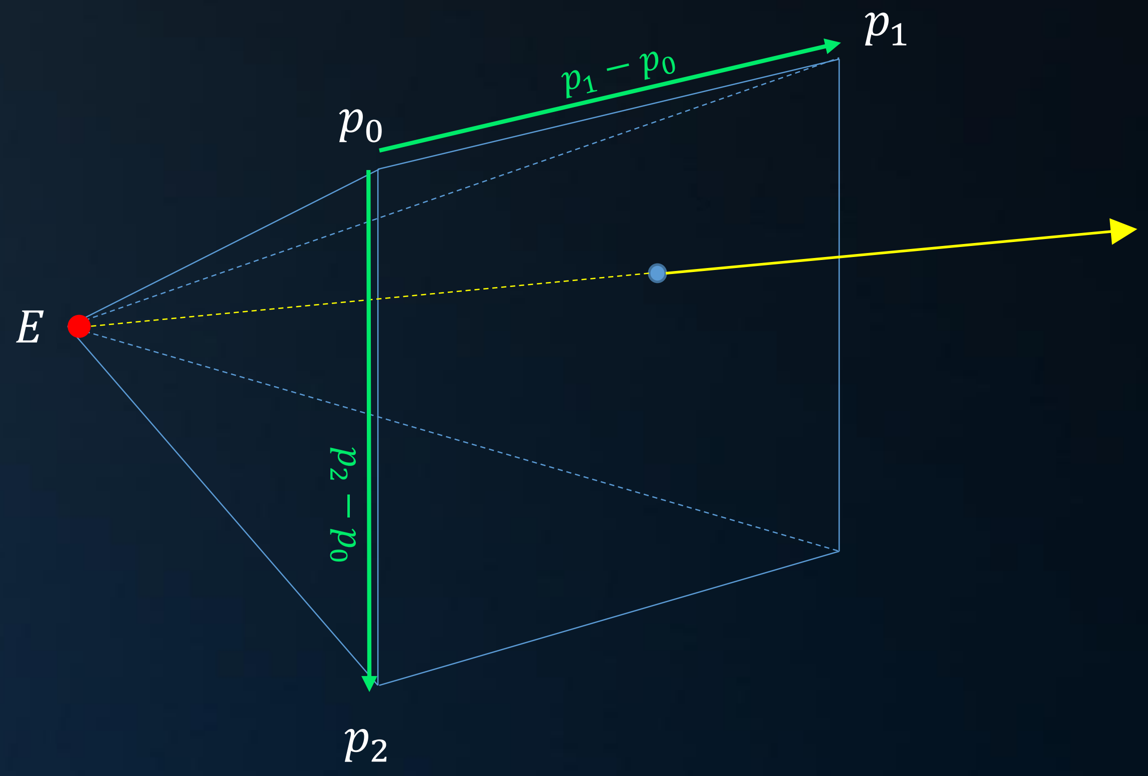
$p(u,v) = p_0 + u(p_1 - p_0) + v(p_2 - p_0)$

Ray direction (before normalization):

$\vec{D} = p(u,v) - E$

Ray origin:

$O = E$



Intersect

Ray intersection

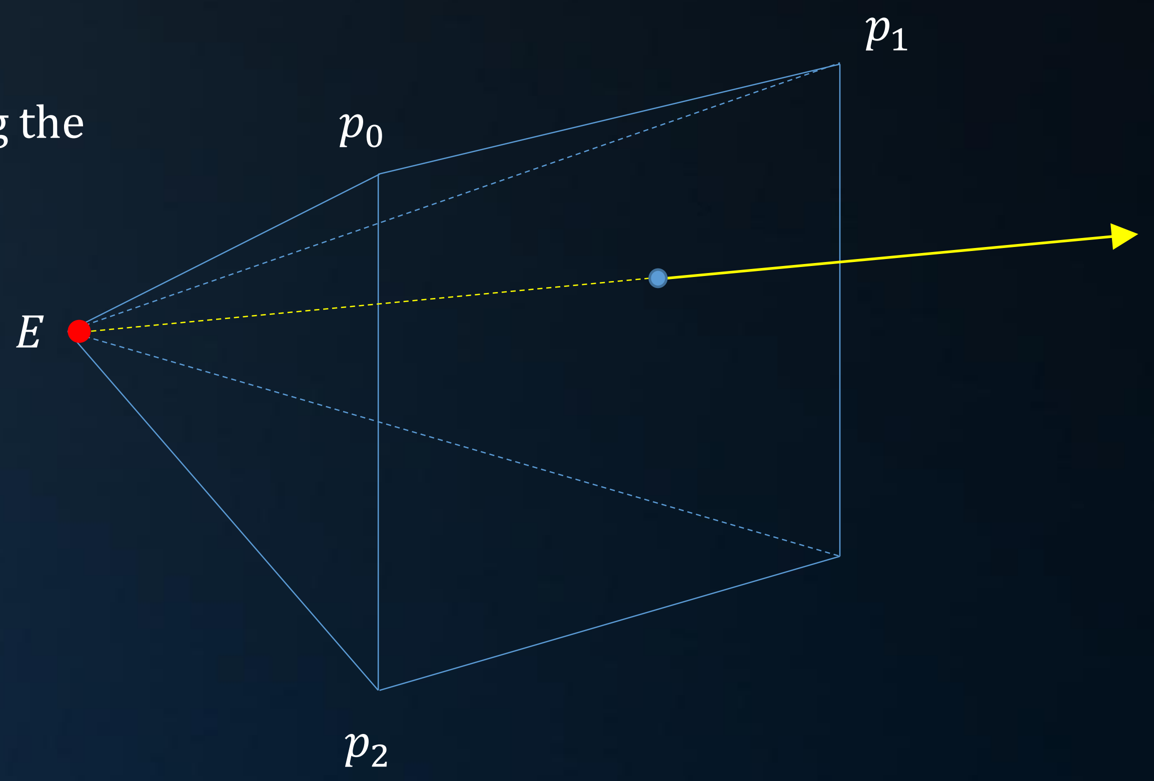
Given a ray $p(t) = O + t\vec{D}$, we determine the smallest intersection distance t by intersecting the ray with each of the primitives in the scene.

Ray / plane intersection:

Plane: $\mathrm{p} \cdot \vec{N} + d = 0$
Ray: $p(t) = O + t\vec{D}$

Substituting for $p(t)$, we get

$$\left(O + t\vec{D}\right) \cdot \vec{N} + d = 0$$
$$t = -(O \cdot \vec{N} + d)/(\vec{D} \cdot \vec{N})$$
$$P = O + t\vec{D}$$



Intersect

Ray intersection

Ray / sphere intersection:

Sphere: $(p - C) \cdot (p - C) - r^2 = 0$
Ray: $p(t) = O + t\vec{D}$

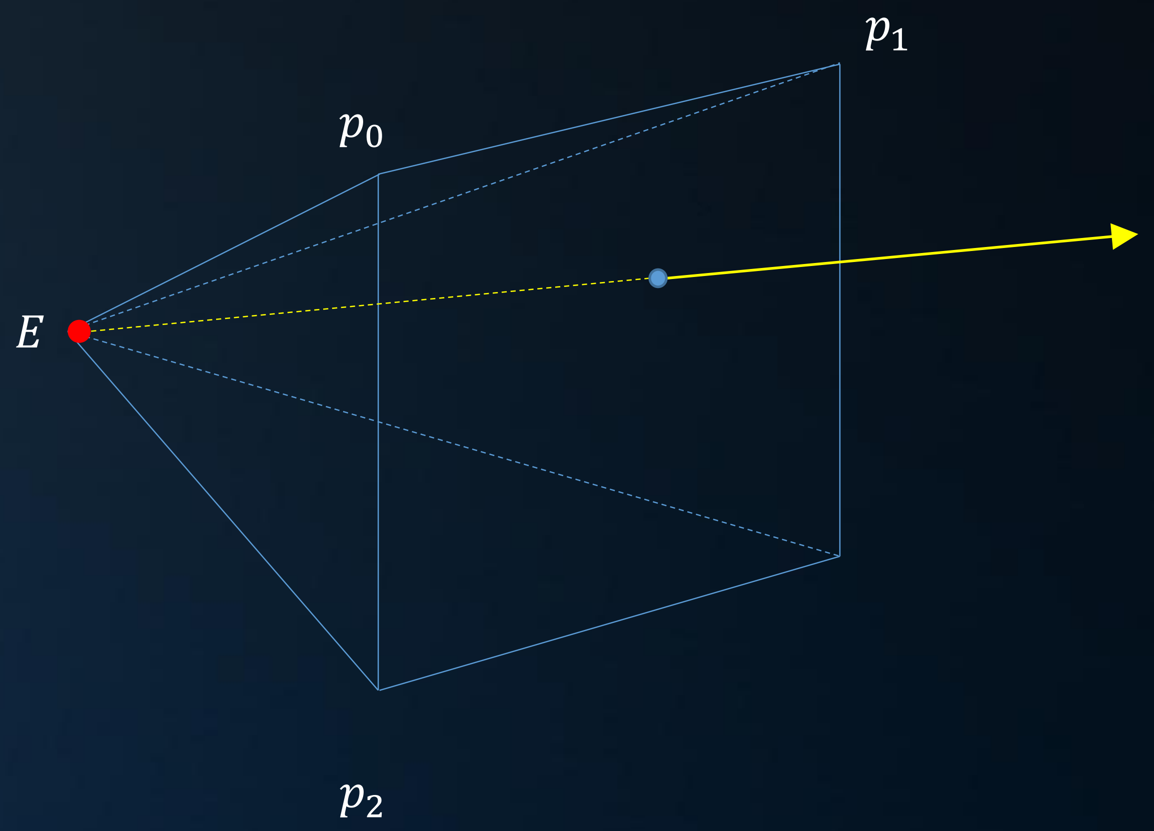
Substituting for $p(t)$, we get

$$\left(O + t\vec{D} - C\right) \cdot \left(O + t\vec{D} - C\right) - r^2 = 0$$
$$\vec{D} \cdot \vec{D}\; t^2 + 2\vec{D} \cdot (O - C)\; t + (O - C)^2 - r^2 = 0$$

$$ax^2 + bx + c = 0 \;\rightarrow\; x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Negative:
no intersections

$a = \vec{D} \cdot \vec{D}$
$b = 2\vec{D} \cdot (O - C)$
$c = (O - C) \cdot (O - C) - r^2$

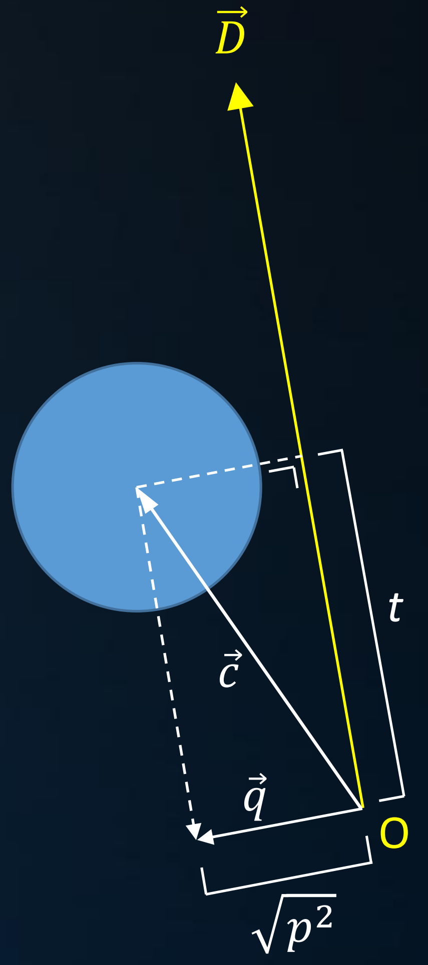


Intersect

Ray Intersection

Efficient ray / sphere intersection:

```
void Sphere::IntersectSphere( Ray ray )
{
    vec3 c = this.pos - ray.O;
    float t = dot( c, ray.D );
    vec3 q = c - t * ray.D;
    float p2 = dot( q, q );
    if (p2 > sphere.r2) return;
    t -= sqrt( sphere.r2 – p2 );
    if ((t < ray.t) && (t > 0)) ray.t = t;
    // or: ray.t = min( ray.t, max( 0, t ) );
}
```



Note:

This only works for rays that start outside the sphere.

Today's Agenda:

- Primitives *(contd.)*
- Ray Tracing
- Intersections
- Assignment 2
- Textures

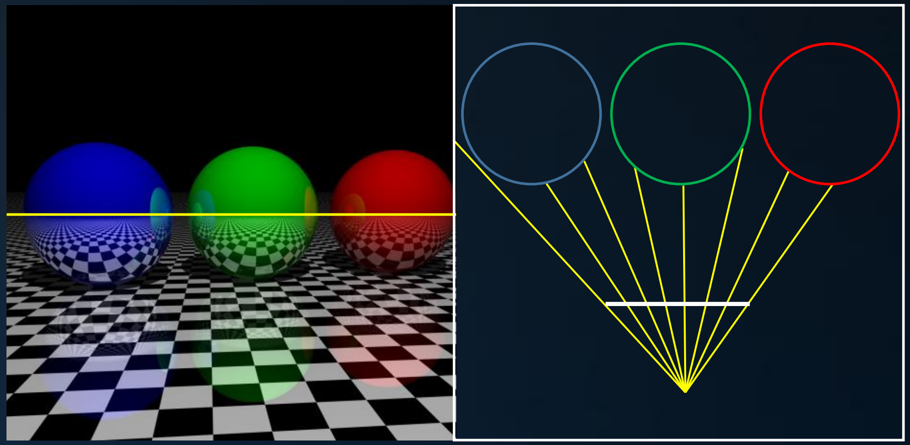
Assignment 2

Deadline assignment 1:

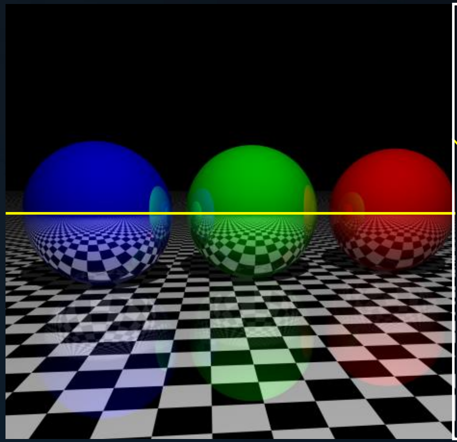
Wednesday May 11, 23.59

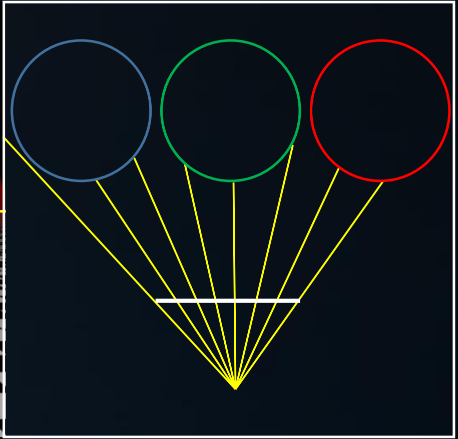
Assignment 2: *”Write a basic ray tracer.”*

- Using the template
- In a 1024x512 window
- Two views, each 512x512
- Left view: 3D
- Right view: 2D slice



Assignment 2





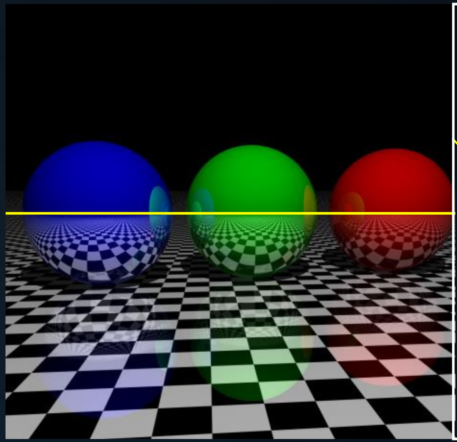
Assignment 2: *”Write a basic ray tracer.”*

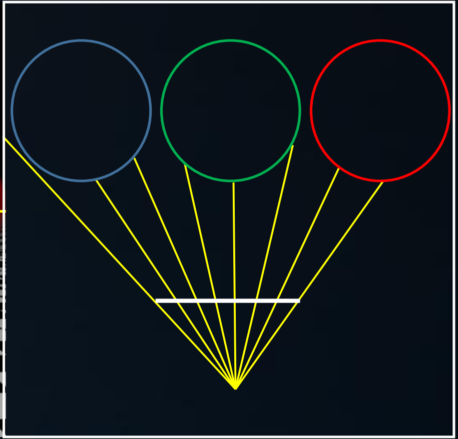
Steps:

1. Create a Camera class; default: position (0,0,0), looking at (0,0,-1).
2. Create a Ray class
3. Create a Primitive class and derive from it a Sphere and a Plane class
4. Add code to the Camera class to create a primary ray for each pixel → *For $y = 0$, visualize every 10th ray*
5. Implement Intersect methods for the primitives
6. Per pixel, find the nearest intersection and plot a pixel → *Visualize the intersection points*
7. Add controls to move and rotate the camera
8. Add a checkerboard pattern to the floor plane.

9. *Add reflections and shadow rays (next lecture).*

Assignment 2

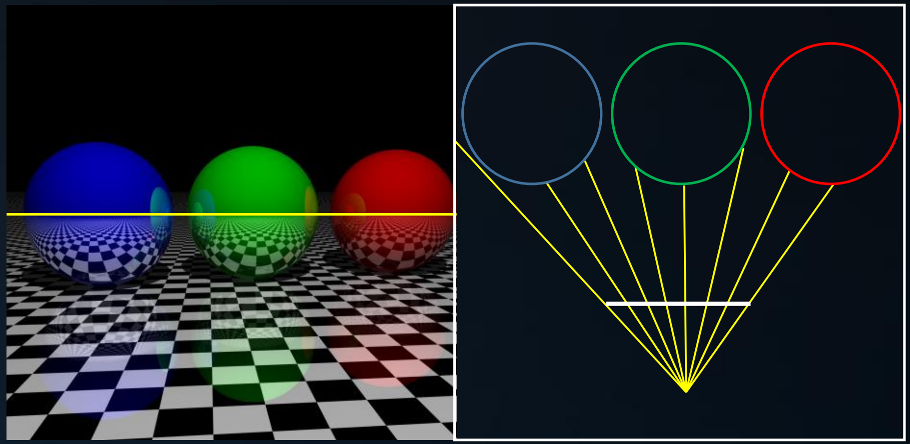




Extra points:

- Add additional primitives, e.g.:
  - Triangle, quad, box
  - Torus, cylinder
  - Fractal
- Add textures to all primitives
- Add a sky dome
- Add refraction and absorption (next lecture)
- One extra point for the fastest ray tracer
- One extra point for the smallest ray tracer meeting the minimum requirements.

Assignment 2



Official:

- Full details in the official assignment 2 document, available today from the website.
- Deadline: May 31st 2016, 23:59.
- Small exhibition of noteworthy entries on Thursday June 2nd and on the website.

Today's Agenda:

- Primitives *(contd.)*
- Ray Tracing
- Intersections
- Assignment 2
- Textures

Textures

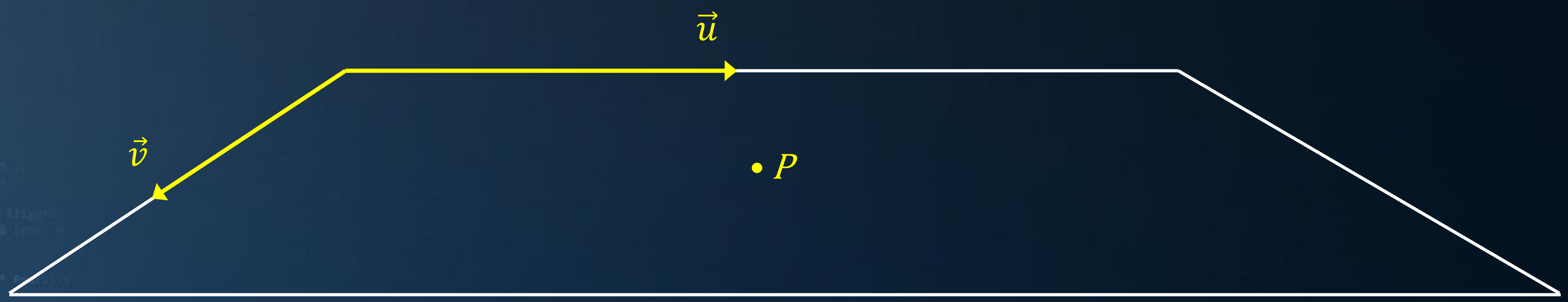
Texturing a Plane

Given a plane: $y = 0$ (i.e., with a normal vector $(0,1,0)$).

Two vectors on the plane define a basis: $\vec{u} = (1,0,0)$ and $\vec{v} = (0,0,1)$.

Using these vectors, any point on the plane can be reached: $P = \lambda_1\vec{u} + \lambda_2\vec{v}$.

We can now use λ_1, λ_2 to define a color at P: $F(\lambda_1, \lambda_2) = \cdots$.



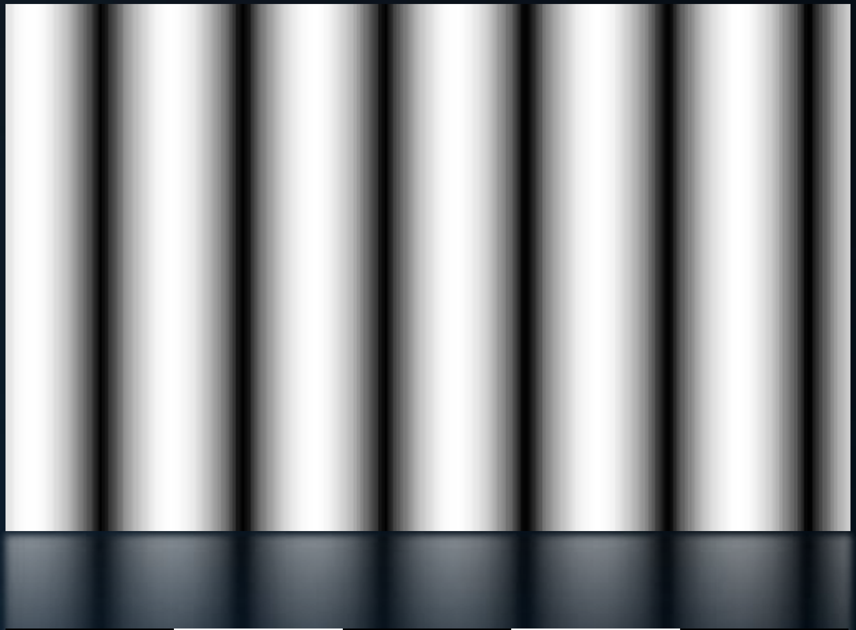
Textures

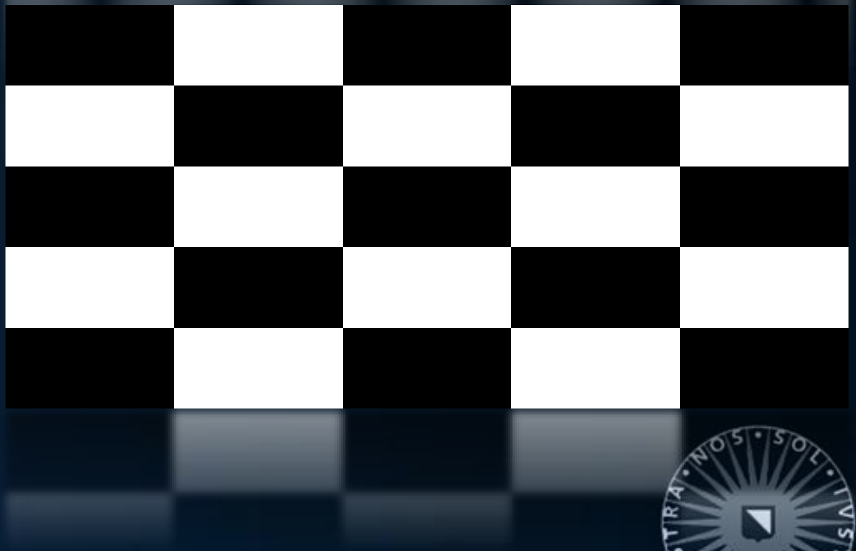
Example:

$$F(\lambda_1, \lambda_2) = \sin(\lambda_1)$$

Another example:

$$F(\lambda_1, \lambda_2) = ((\text{int})(2 * \lambda_1) + (\text{int})\lambda_2)\ \&\ 1$$



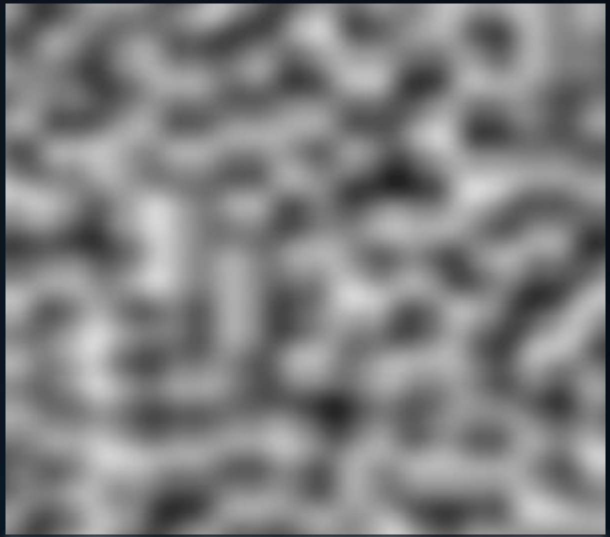


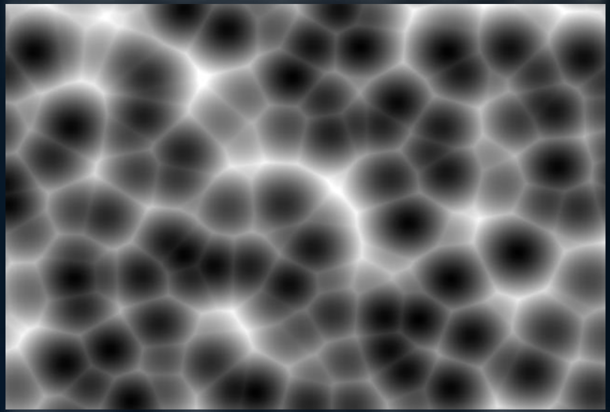
Textures

Other examples (not explained here):

Perlin noise
Details: http://www.noisemachine.com/talk1

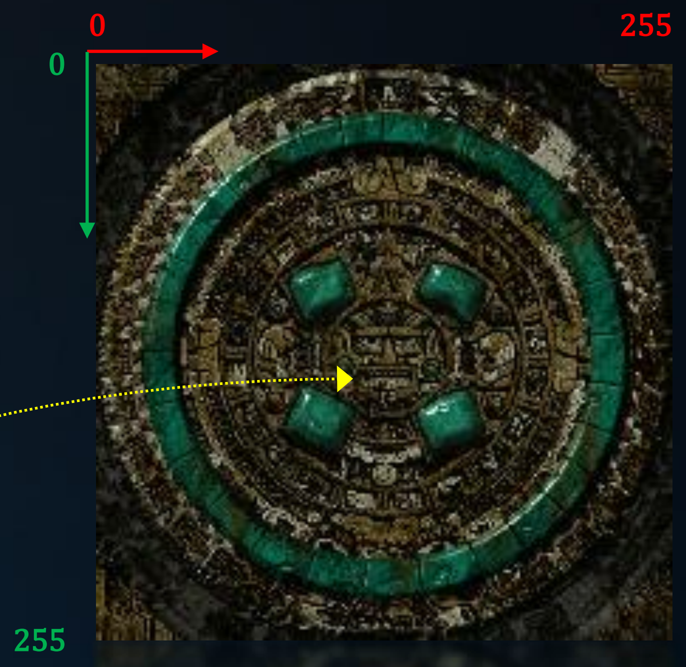
Voronoi / Worley noise
Details: "A cellular texture basis function", S. Worley, 1996.






life: 100
ammo: 050

Textures

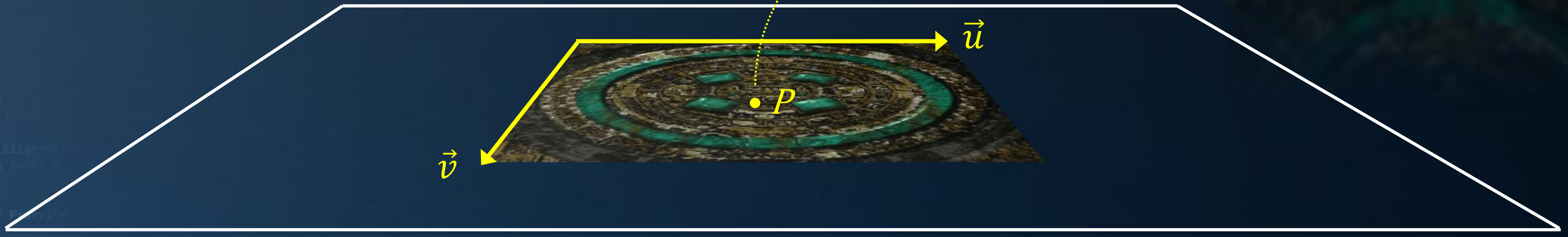


Obviously, not all textures can be generated procedurally.

For the generic case, we lookup the color value in a pixel buffer.

$$\binom{x}{y} = \binom{P \cdot \vec{u}}{P \cdot \vec{v}} * \binom{T_{width}}{T_{height}}$$

Note that we find the pixel to read based on P; we don't find a 'P' for every pixel of the texture.



Textures

Retrieving a pixel from a texture:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} P \cdot \vec{u} \\ P \cdot \vec{v} \end{pmatrix} * \begin{pmatrix} T_{width} \\ T_{height} \end{pmatrix}$$

We don’t want to read outside the texture. To prevent this, we have two options:

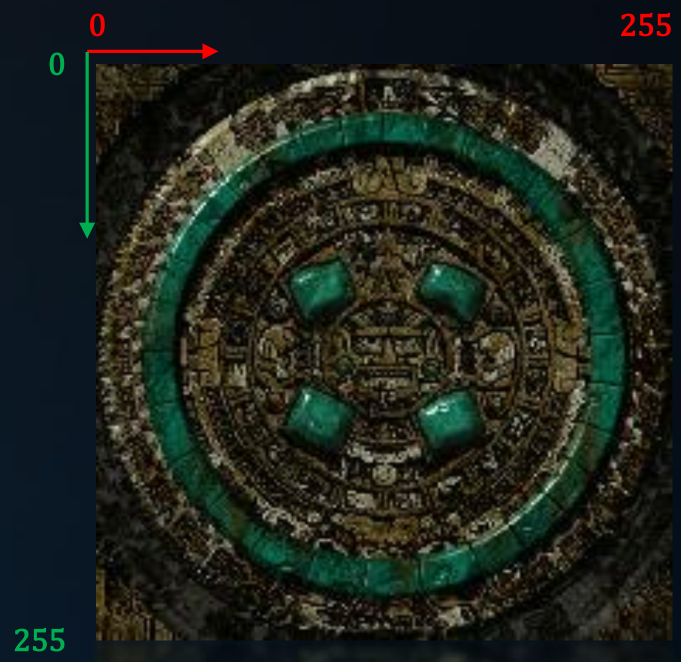
1. Clamping

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} clamp(P \cdot \vec{u}, 0, 1) \\ clamp(P \cdot \vec{v}, 0,1) \end{pmatrix} * \begin{pmatrix} T_{width} \\ T_{height} \end{pmatrix}$$

2. Tiling

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} frac(P \cdot \vec{u}) \\ frac(P \cdot \vec{v}) \end{pmatrix} * \begin{pmatrix} T_{width} \\ T_{height} \end{pmatrix}$$

Tiling is efficiently achieved using a bitmask. This requires texture dimensions that are a power of 2.

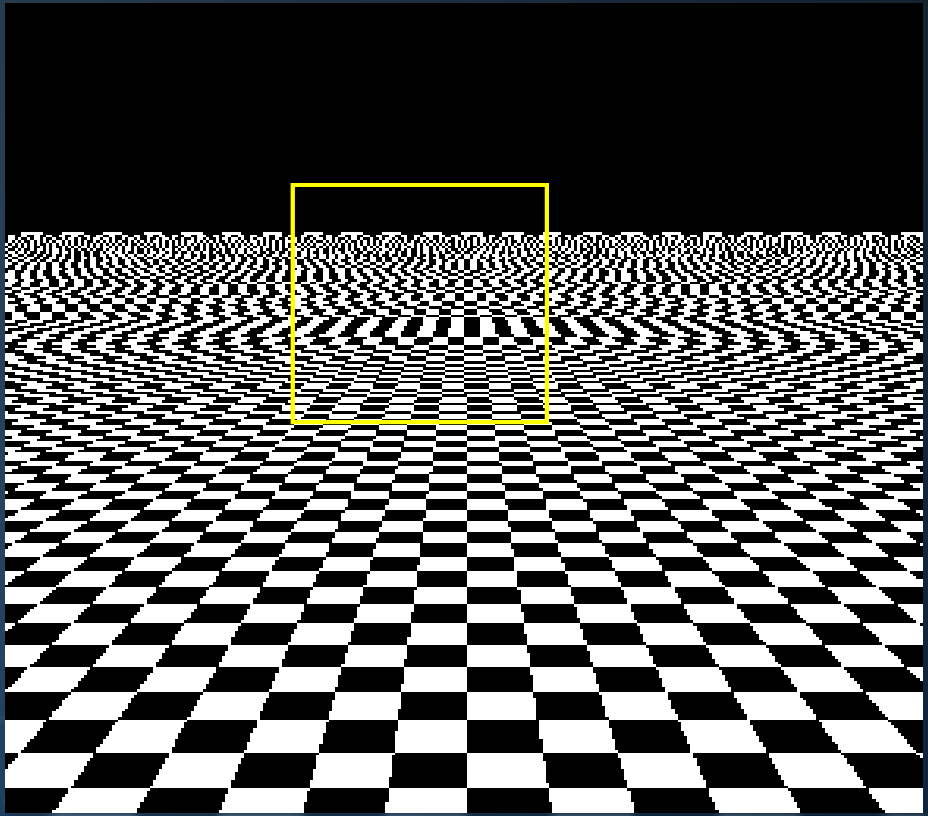


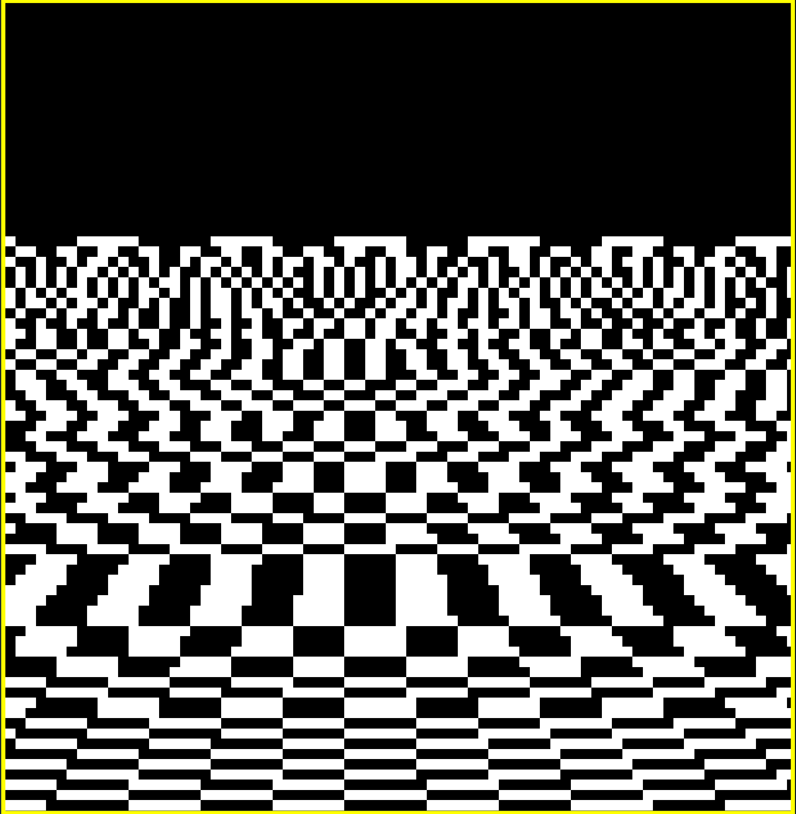
Textures

Texture mapping: oversampling

Textures

Texture mapping: undersampling





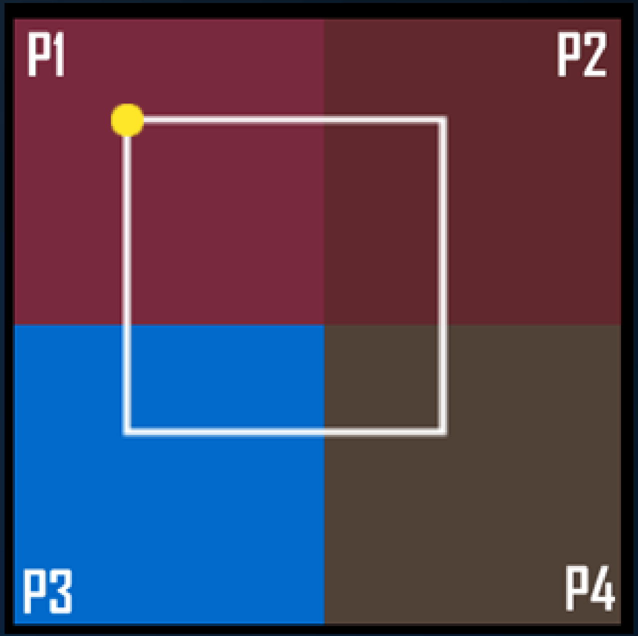
Textures

Fixing oversampling

Oversampling: reading the same pixel from a texture multiple times.
Symptoms: blocky textures.

Remedy: bilinear interpolation:
Instead of clamping the pixel location to the nearest pixel, we read from four pixels.

$$w_{p1}: (1 - frac(x)) * (1 - frac(y))$$
$$w_{p2}: frac(x) * (1 - frac(y))$$
$$w_{p3}: (1 - frac(x)) * frac(y)$$
$$w_{p4}: 1 - (wP_1 + wP_2 + wP_3)$$



Textures

Fixing oversampling

Textures

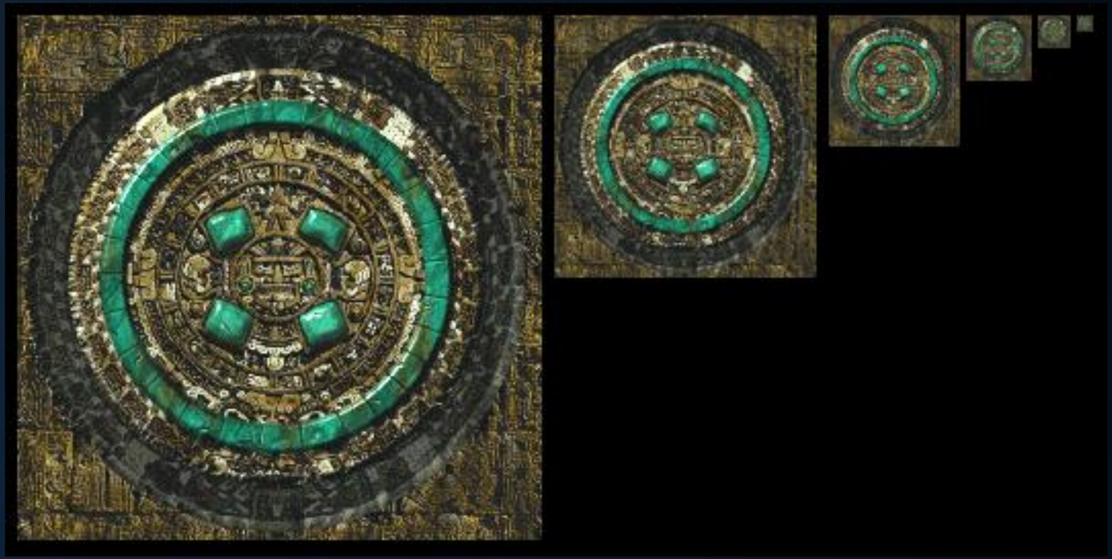
Fixing undersampling

Undersampling: skipping pixels while reading from a texture.
Symptoms: Moiré, flickering, noise.

Remedy: MIP-mapping.

The texture is reduced to 25% by averaging 2x2 pixels. This is repeated until a 1x1 image remains.

When undersampling occurs, we switch to the next MIP level.




NO MIPMAPS


WITH MIPMAPS

Textures

Trilinear interpolation: blending between MIP levels.

Today's Agenda:

- Primitives *(contd.)*
- Ray Tracing
- Intersections
- Assignment 2
- Textures

INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 3: “Ray Tracing (Introduction)”

END of “Ray Tracing (Introduction)”

next lecture: “Ray Tracing (Part 2)”