CPSC 427
Video Game Programming

Rendering and Transformations

Helge Rhodin

Today

- *ECS summary*
- *2D Transformations*
- *Some graphics*

- *Your pitches*

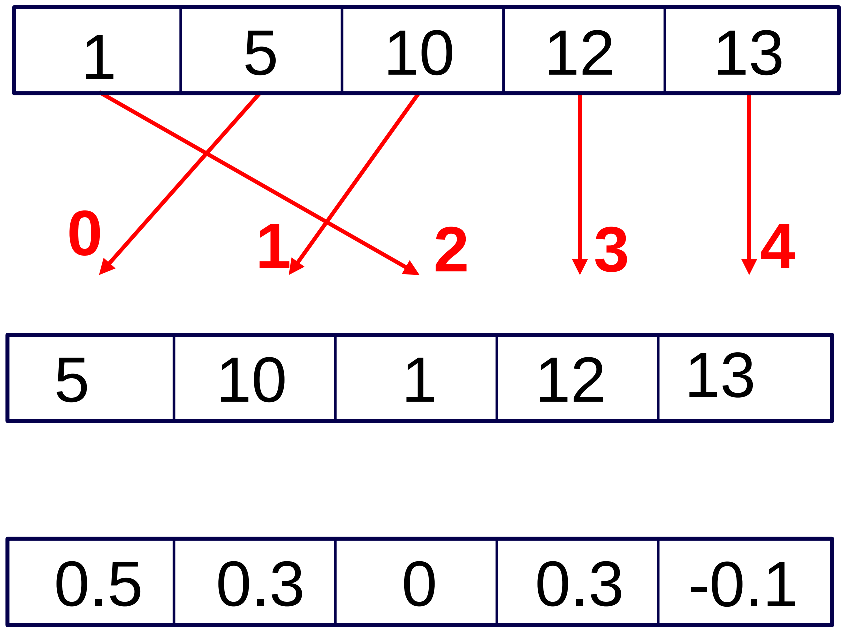
Map + Dense Array (example)

Registry for one component

map: Entity IDs (keys) → Array indices

entities array (component values)

components array (component values)



Iterate over all velocity components that belong to an entity with a position

```
for(Entity entity : registry<Velocity>.entities) // using the key array
    if (map<Position>.has(entity)) // using the map
        map<Position>.get(entity)+= registry<Velocity>.get(entity); // using the map
```

Faster iteration via entity and component array

Accessing the velocity map (map<Velocity>) is an unnecessary indirection

```
for(Entity entity : entities<Velocity>)
    if (map<Position>.has(entity))
        map< Position >.get(entity) += map<Velocity>.get(entity);
```

We can access the velocity components in linear fashion

```
for(int vi = 0; vi < entities<Velocity>.size(); vi++)
    Entity entity : entities<Velocity>[vi];
    pi = map<Position>.get(entity);
    if (pi)
        components< Position >[pi] += components< Velocity >[vi];
```

Self study:
The Sparse Map

Used by EnTT GAMING MEETS MODERN C++
https://github.com/skypjack/entt

	ID	Index Pos	Index Vel	Index Jump	Index Player	Index Squish
Mario	1			1	1	
Goomba1	2					1
Luigi	3			2		
Goomba2	4					2

Sparse array (ID -> dense index)

Position	Velocity	Jumps	Player	Squishable

Dense component arrays

Issues?

Concept: Sparse array + dense array

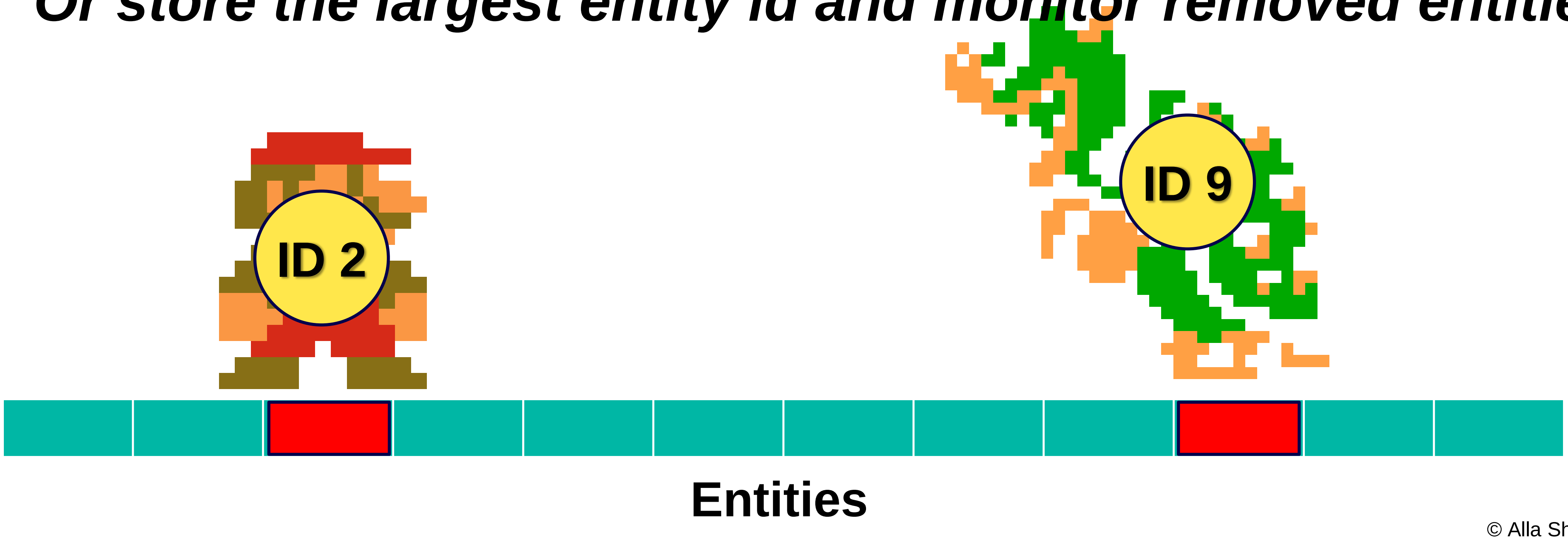
Implementation: std:vector<Entity> entities; std:vector<unsigned int> indices; std:vector<Components> components;

Deletion of components

- **When we "delete" an entity we must delete corresponding components to.**
- **Different approaches to this,**
  - ***Fill deleted components in arrays with the last entities data***
    - Extra care must be taken when managing indices
  - ***Mark spots in arrays as rewritable***
    - Big systems will suffer from poor memory management

Lifetime of entities

- Each Entity is typically just a unique identifier to its components
- Store Entities in a big static array in the Entity Manager
  - *Or store the largest entity id and monitor removed entities*



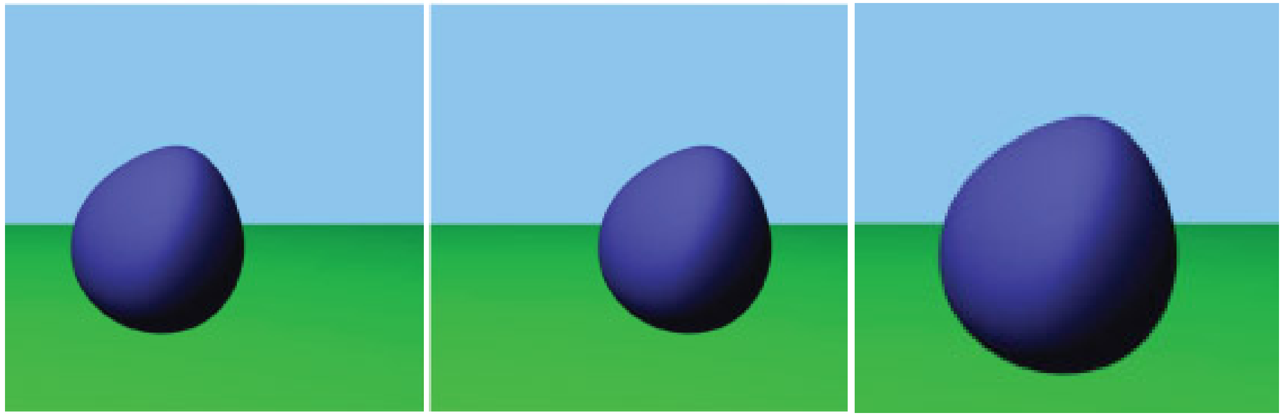
How Does a System Find its Entities?

Extension/Optimization:

- **Each system has a list of entity IDs it is interested in**
- **Systems register their bitsets/bitmaps with the Entity Manager**
- **Whenever an Entity is added…**
  - ***Evaluate which systems are interested & update their ID lists***

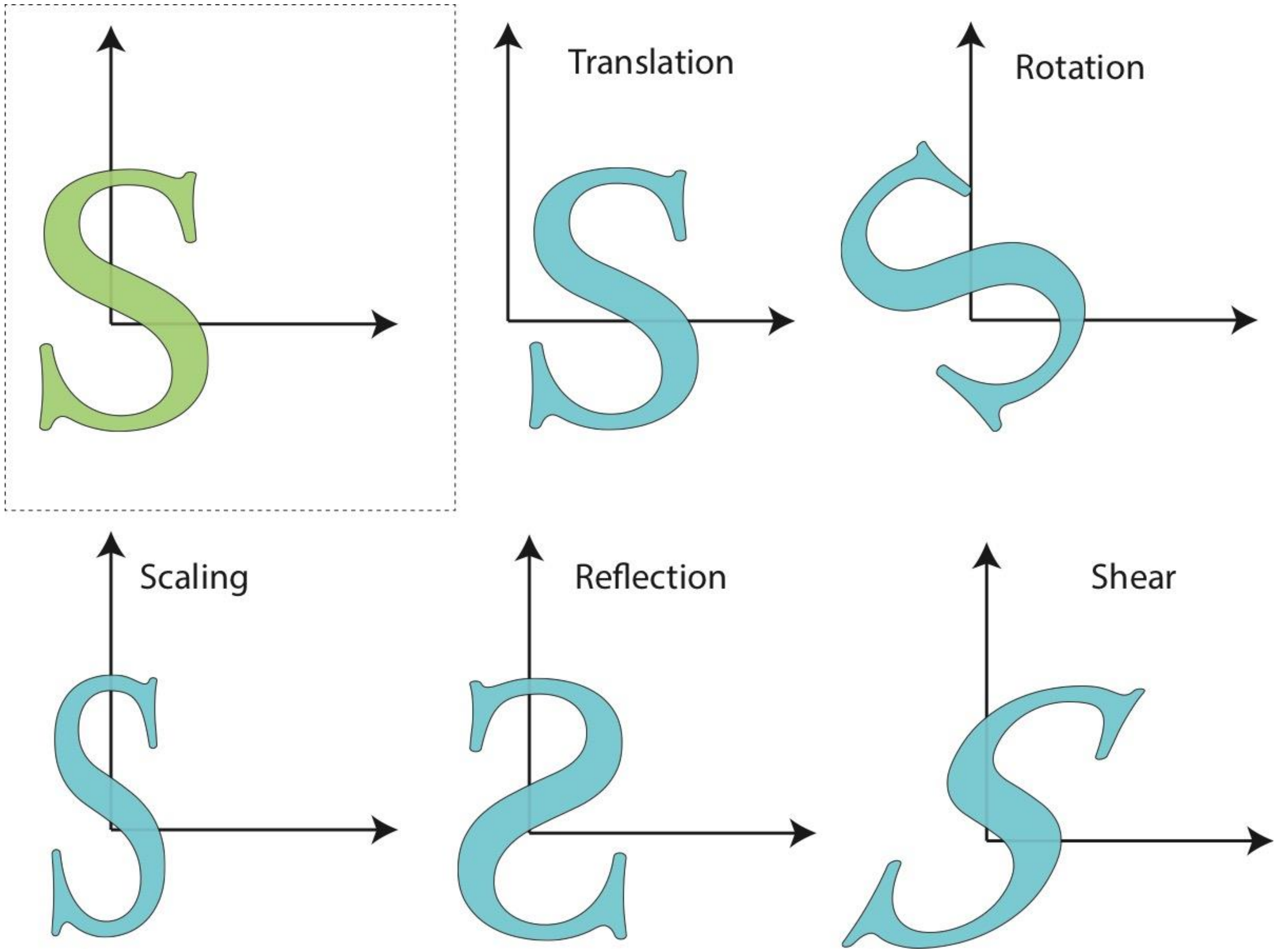
CPSC 427
Video Game Programming

Transformations



Helge Rhodin

Modeling Transformations



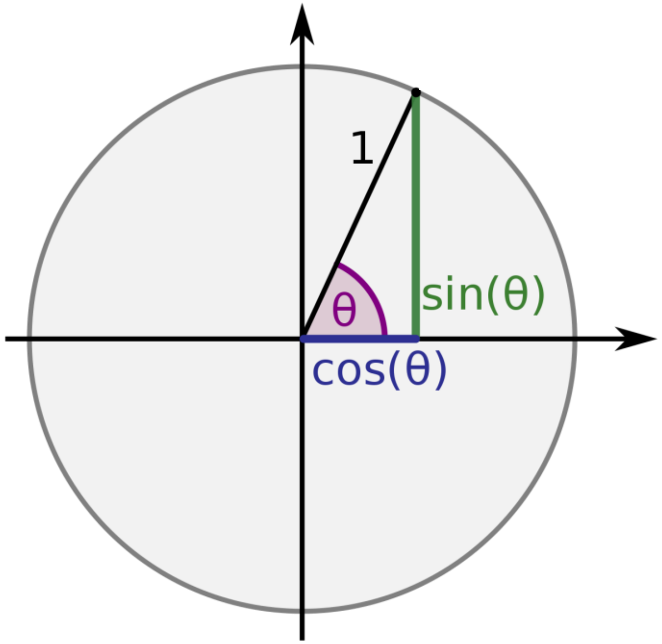
Linear transformations

- Rotations, scaling, shearing
- Can be expressed as 2x2 matrix (for 2D points)
- E.g.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- or a rotation

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$



Rotation angle θ, cos, and sin

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

Affine transformations

- Linear transformations + translations
- Can be expressed as 2x2 matrix + 2 vector
- E.g. scale+ translation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Modeling Transformation

Adding third coordinate

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \Rightarrow \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 0 & t_x \\ 0 & 2 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Affine transformation are now linear

- one 3x3 matrix can express: 2D rotation, scale, shear, and translation

Self study: Homogeneous coordinates

- Homogeneous coordinates are defined as vectors, with equivalence

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix} = \begin{pmatrix} x\lambda \\ y\lambda \\ z\lambda \end{pmatrix}$$

- Can also represent projective equations
- 3x3 homogeneous matrix becomes 4x4

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 & t_x \\ 0 & 2 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

CPSC 427
Video Game Programming

Rendering basics

Helge Rhodin

What is rendering?

Generating an image from a (3D) scene

Let’s think how!

Scene

- A coordinate frame
- Objects
- Their materials
- (Lights)
- (Camera)

Object

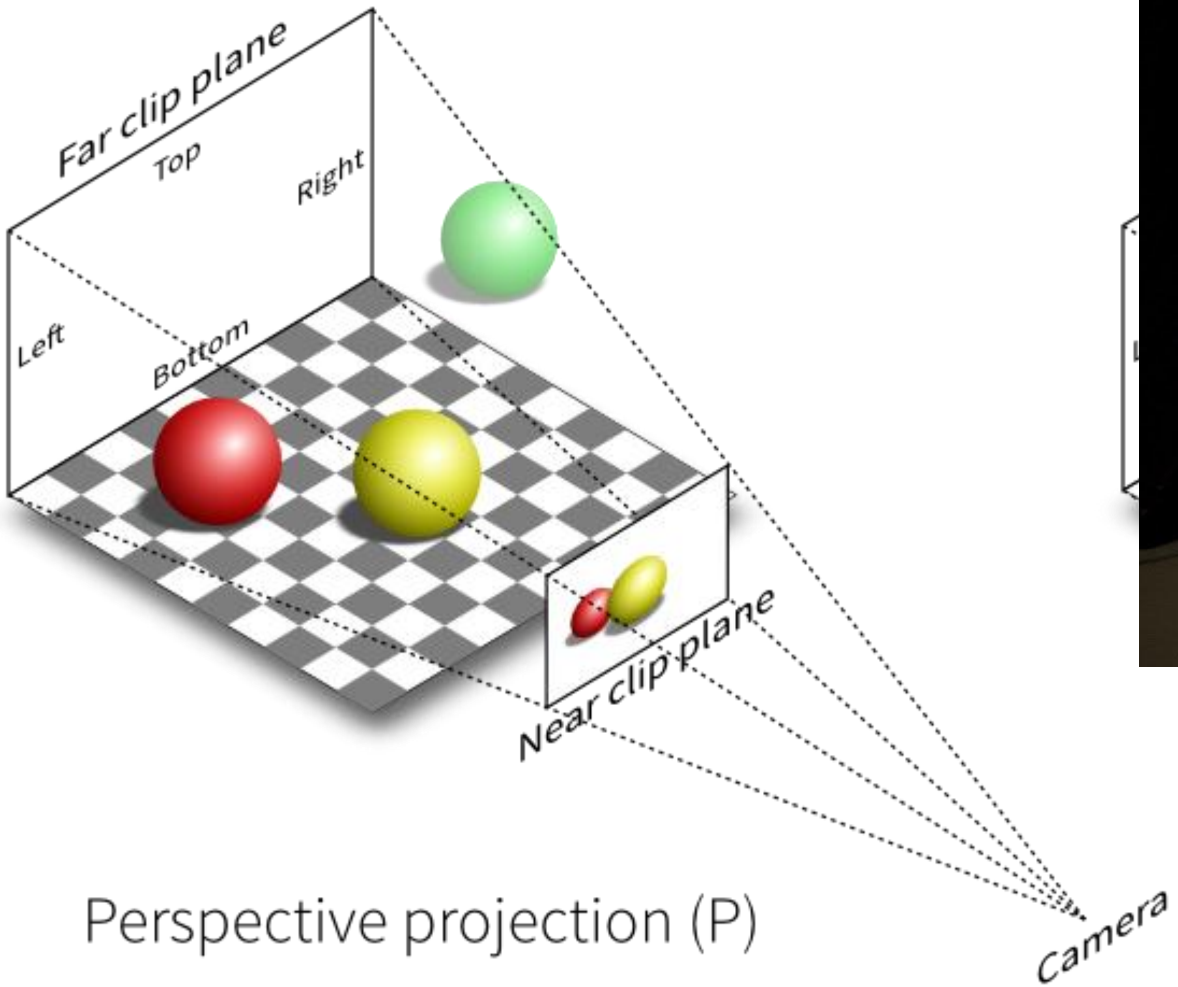
Most common:

- ***surface representation***

Image

A grid of color values

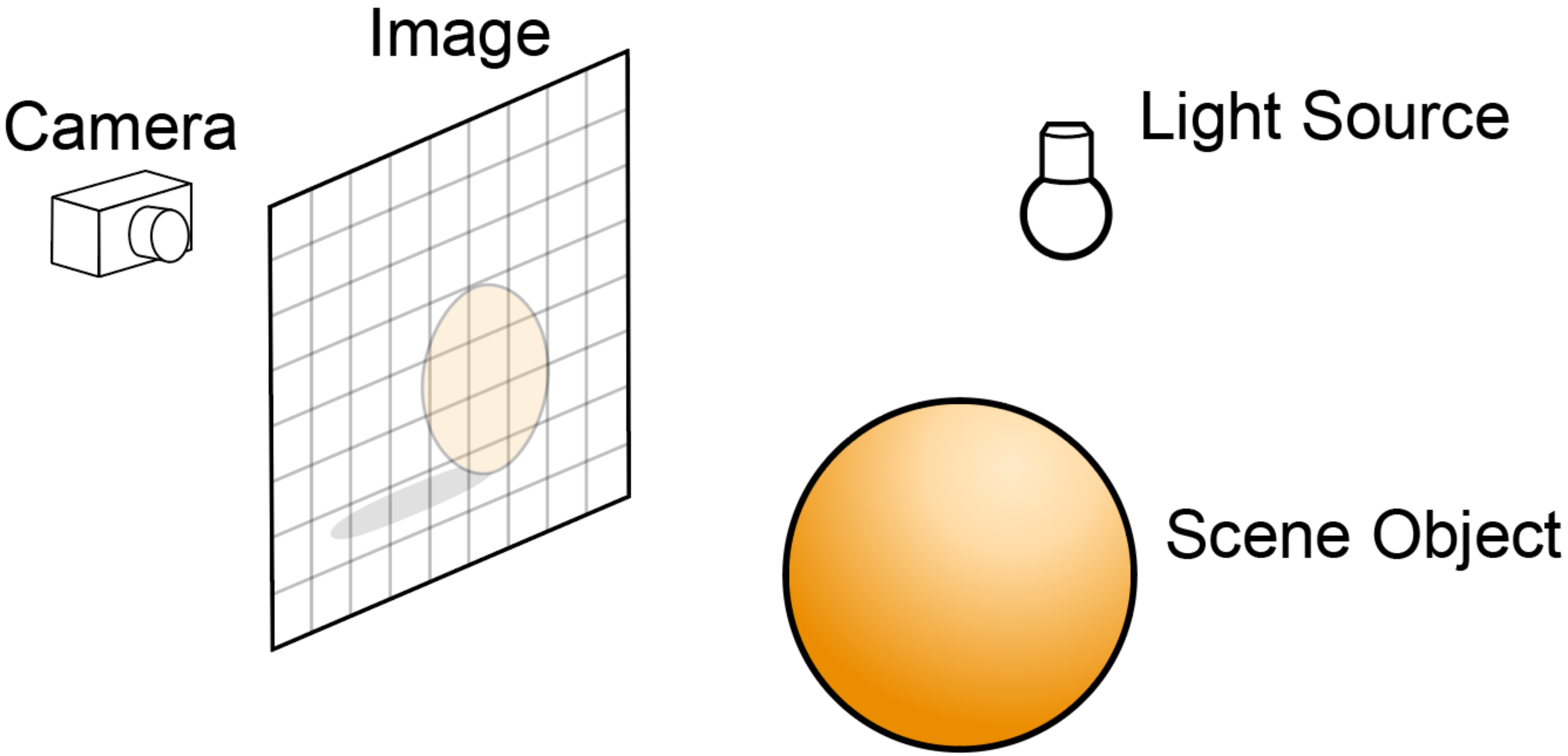
Virtual Camera



Virtual camera registered in the real world
(using marker-based motion capture)

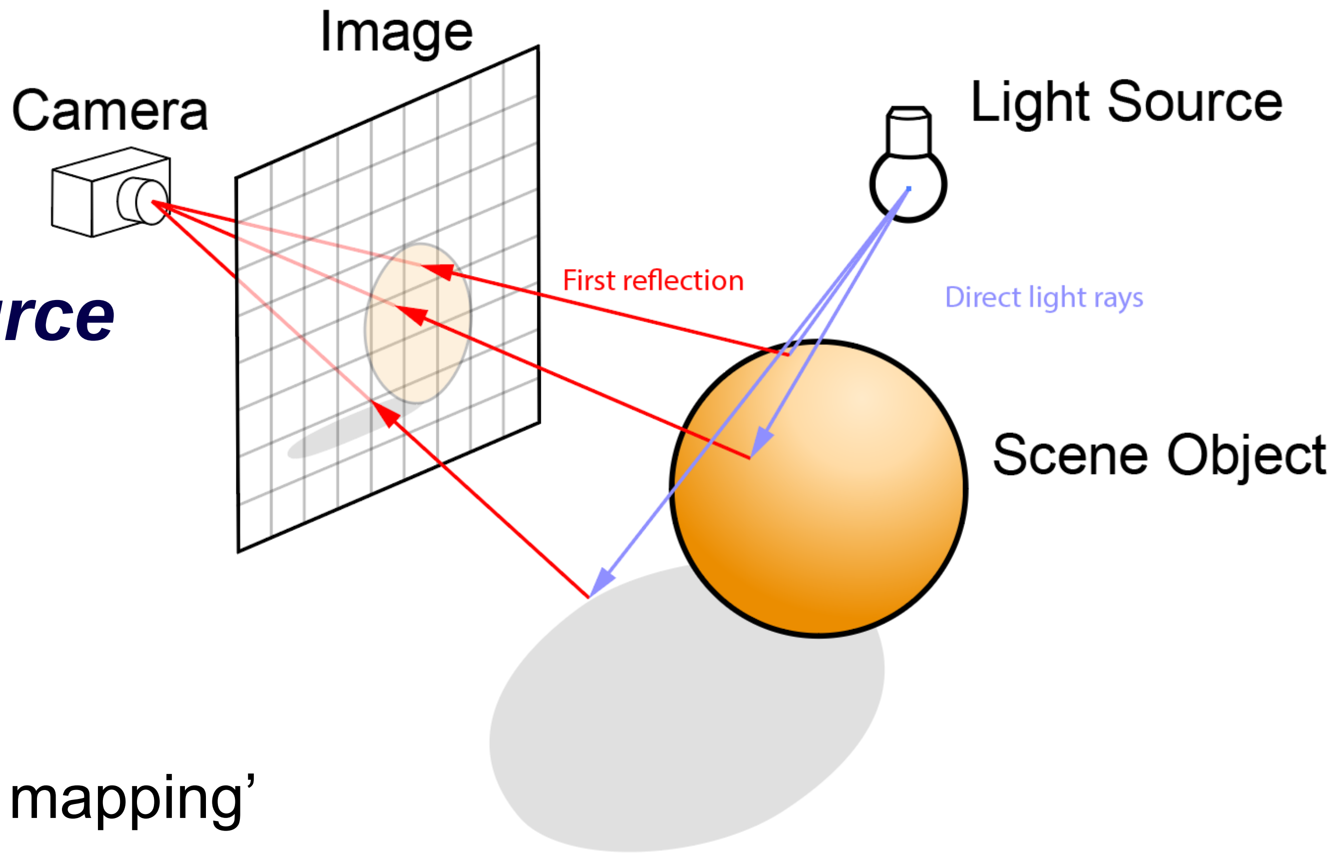
Rendering?

- ***Simulating light transport***
- *How to simulate light efficiently?*



Rendering – ‘Light’ Tracing

- ***simulate physical light transport from a source to the camera***
  - *the paths of photons*



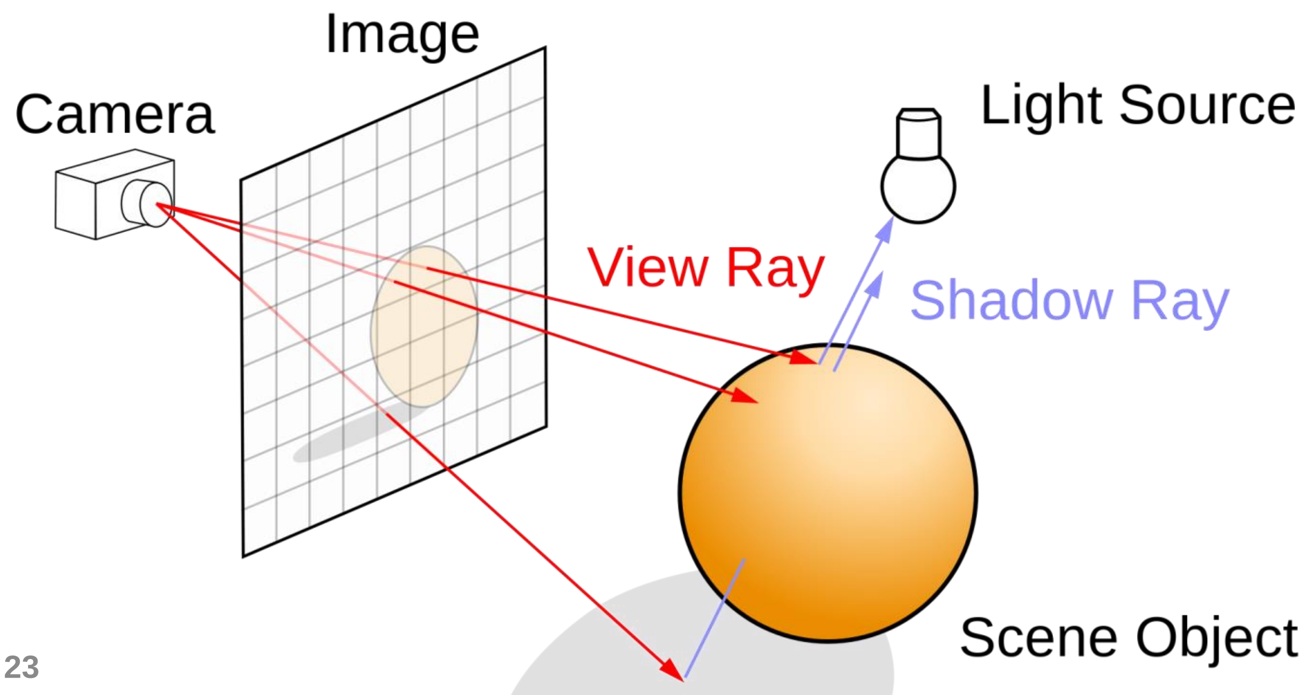
- ***shoot rays from the light source***
  - *random direction*
- ***compute first intersection***
  - *continue towards the camera*

- used for indirect illumination: ‘photon mapping’

Rendering – Ray Tracing

Start rays from the camera (opposes physics, an optimization)

- *View rays: trace from every pixel to the first occlude*
- *Shadow ray: test light visibility*



Nvidia RTX does ray tracing

Problems of ray tracing

- ***the collision detection is costly***
- ray-object intersection
  - *n objects*
  - *k rays*
  - *naïve: O(n*k) complexity*

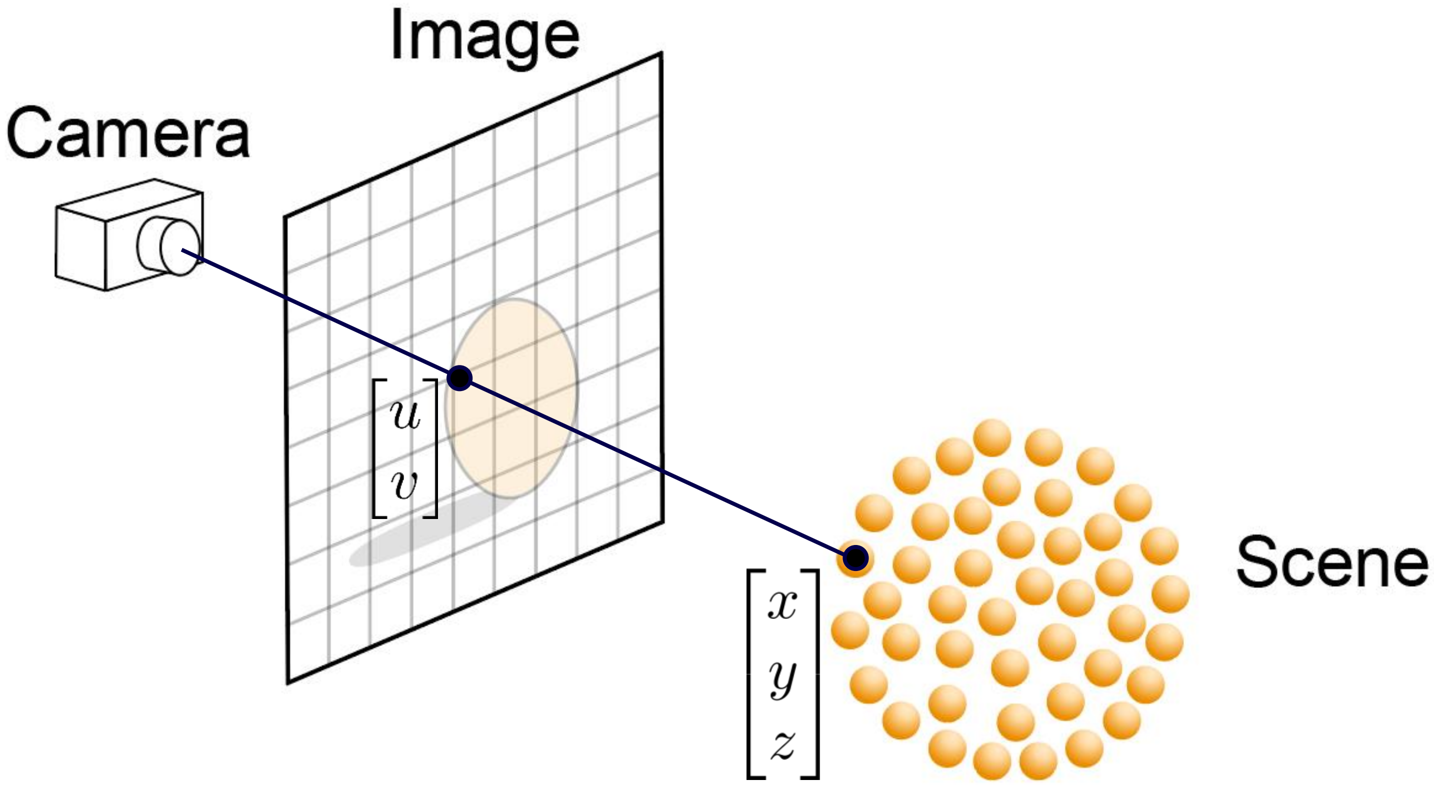
Rendering – Splatting

Approximate scene with spheres

- *sort spheres back-to front*
- *project each sphere*
  - simple equation

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{z} \begin{bmatrix} x \\ y \end{bmatrix}$$

  - O(n) for n spheres



Many spheres needed!

Shadows?

Rendering – Rasterization

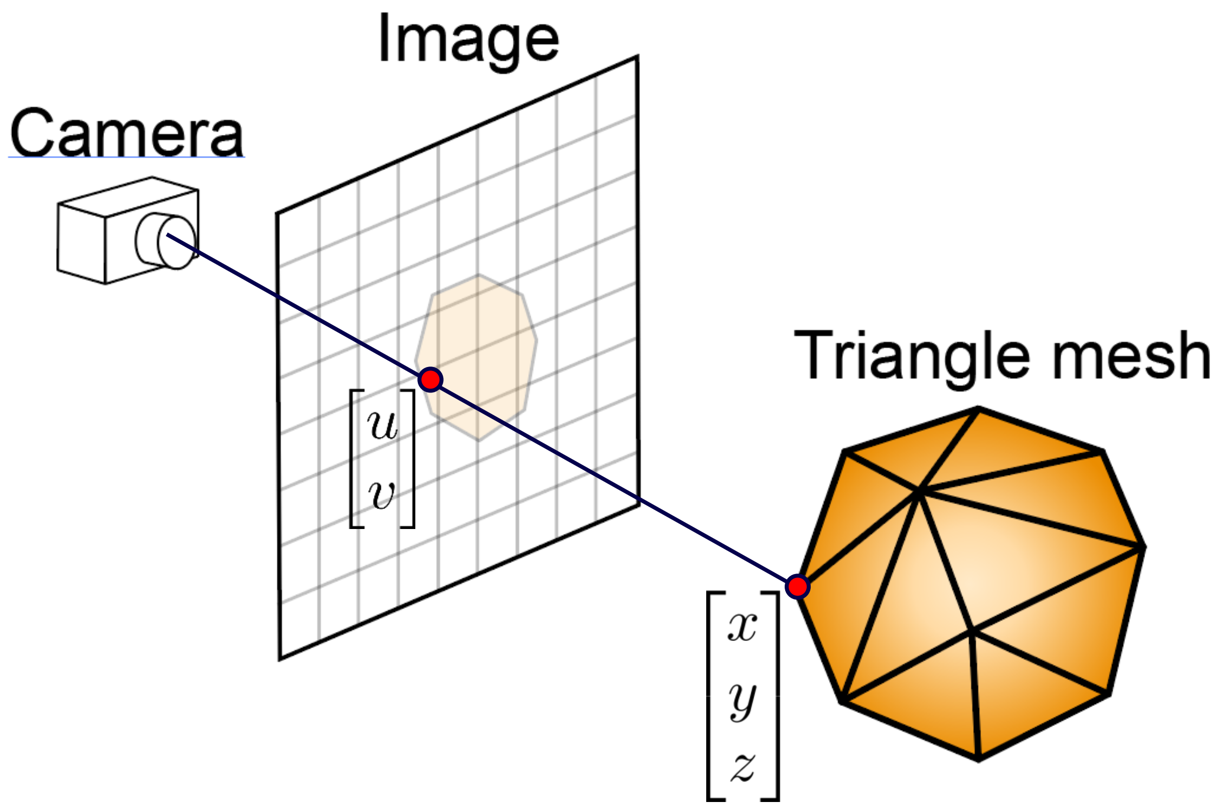
Approximate objects with triangles

1. *Project each corner/vertex*
   - projection of triangle stays a triangle

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{z} \begin{bmatrix} x \\ y \end{bmatrix}$$

   - O(n) for n vertices

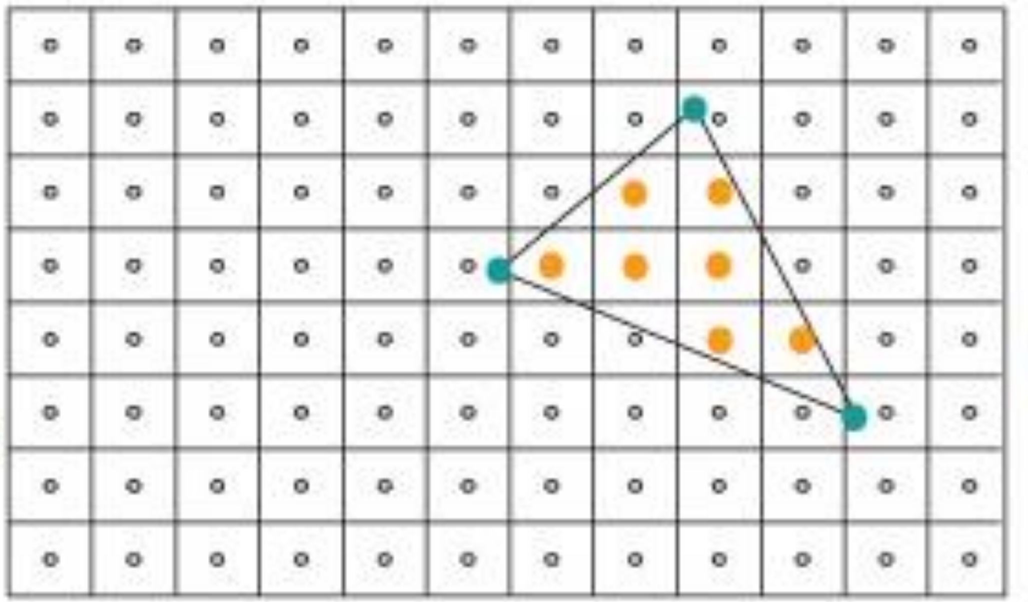
2. *Fill pixels enclosed by triangle*
   - e.g., scan-line algorithm



Rasterizing a Triangle

- ***Determine pixels enclosed by the triangle***
- ***Interpolate vertex properties linearly***

Vertices



Fragments

(for every pixel; color or attributes to compute color: texture coordinate, direction, …)

Self study:
Interpolation with barycentric coordinates

- *linear combination of vertex properties*
  - *e.g., color, texture coordinate, surface normal/direction*

- *weights are proportional to the areas spanned by the sides to query point P*