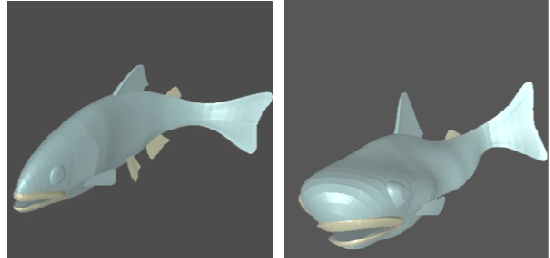


## Adding realism

### Overview of further methods

Warping and free-form deformation  
Ray tracing  
Particle systems

## Warping and free-form deformation

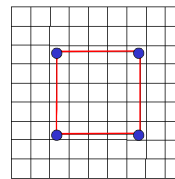


[http://lgm.fri.uni-lj.si/RG/KRIVLJENJE\\_3D/3D\\_krivljenje.ppt](http://lgm.fri.uni-lj.si/RG/KRIVLJENJE_3D/3D_krivljenje.ppt)

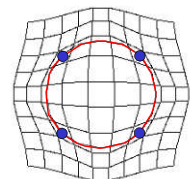
## Warping and free-form deformation

- Free-Form Deformation (FFD) is a technique for modelling 3D deformable objects
- FFDs are defined by a mesh of control points with uniform spacing
- An underlying object is deformed by manipulating a mesh of control points
  - control point can be displaced from their original location
  - control points provide a parameterization of the transformation

## Warping and free-form deformation



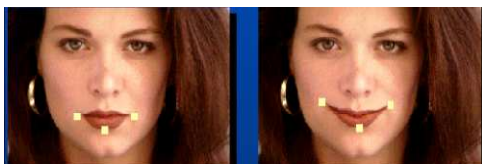
Define a mesh around an object



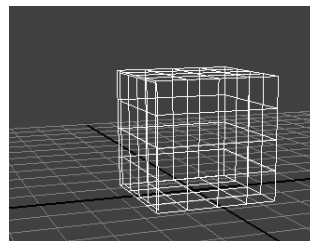
Deform the mesh at all the nodes as required.

Then interpolate the deformation for all the object vertices and pixels

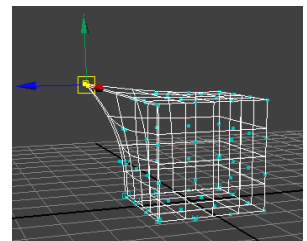
## Warping and free-form deformation 2D



## Warping and free-form deformation 3D



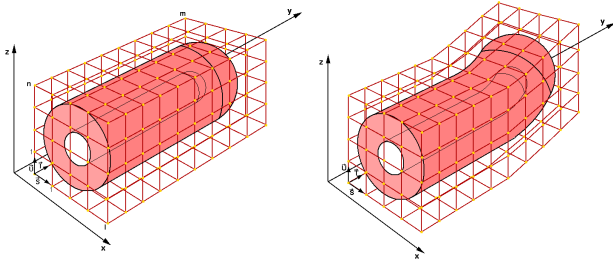
Define a mesh around an object



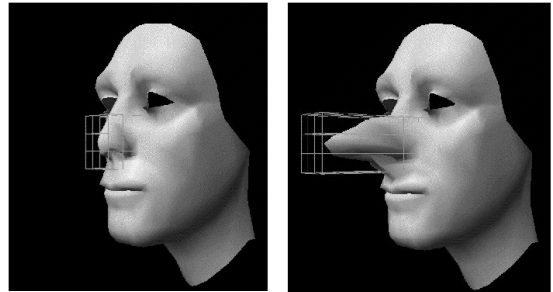
Deform the mesh at all the nodes as required.

Then interpolate the deformation for all the object vertices and pixels (c.f. Gouraud or Phong method)

## Warping and free-form deformation 3D



## Warping and free-form deformation 3D



## Ray tracing

- Ray tracing is one of the most popular methods used in 3D computer graphics to render an image
- It works by tracing the path taken by a ray of light through the scene, and calculating reflection, refraction, or absorption of the ray whenever it intersects an object in the world
- Adds more realism to the rendered scene by allowing effects such as
  - shadows
  - transparency
  - reflections and self-reflections

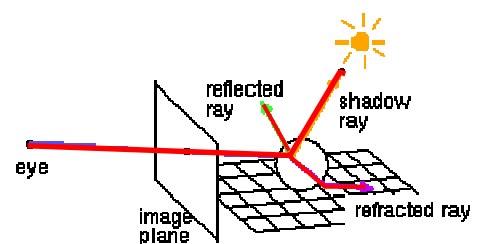


## Ray tracing

### General scheme

- For each pixel on the screen, a ray is defined by the line joining the viewpoint and the pixel of the viewing plane (perspective projection) or by a line orthogonal to the viewing plane
- Each ray is cast through the viewing volume and checked for intersections with the objects inside the viewing volume

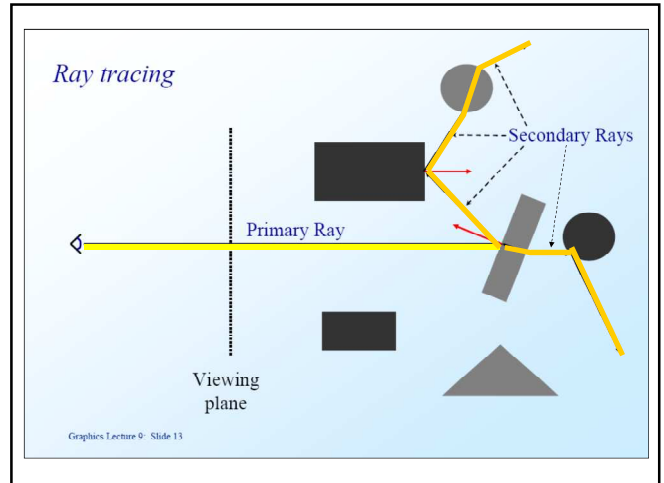
## Ray tracing



## Ray tracing

### Backward ray tracing

- Follow light rays backward from the camera.
- Only one ray for each pixel.
- When (if) the backward ray hits an object, determine the intensity of light and colour coming from the object at the intersection point.



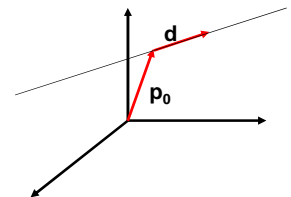
## Ray tracing - calculations

- For each ray we must
  - Calculate all possible intersections with each object inside the viewing volume
  - Find the nearest intersection point
- The scene can be defined using
  - Solid models, e.g.
    - Sphere
    - Cylinder
  - Surface models, e.g.
    - planes
    - triangles
    - polygons
    - Bézier patches

## Ray tracing - calculations

### Rays

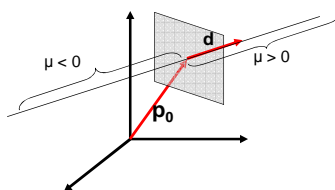
- Rays are parametric lines
- Can be defined by
  - origin  $p_0$
  - direction  $d$
- Equation of ray:
- $p = p_0 + \mu d$



## Ray tracing - calculations

### Calculating Intersection

- The viewing ray can be parameterized by  $\mu$ :
  - $\mu > 0$  denotes the part of the ray behind the viewing plane
  - $\mu < 0$  denotes the part of the ray in front of the viewing plane
- For any visible intersection point  $\mu > 0$



## Ray tracing - calculations

- We now have to calculate the point of intersection of each ray with each surface on its path
- Different surface representations require different solutions. See the website below for equations for sphere, cylinder, plane and triangle

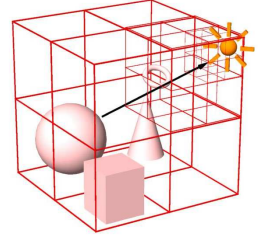
- <http://www.doc.ic.ac.uk/~dfg/graphics/graphics2009/GraphicsLecture10.pdf>
- <http://www.doc.ic.ac.uk/~dfg/graphics/graphics2009/GraphicsLecture11.pdf>

## Ray tracing – speeding up the calculations

- Ray tracing is computationally very demanding
- Various methods exist to speed up the calculations
- The most common are
  - Finding bounding boxes (2D)
  - Finding bounding volumes (3D)
  - Limiting the number of interactions that any ray has with objects in the scene
- We first determine “chunks” of image or space which contains object projections (2D) or objects (3D)
- We only “fire” rays at the non-empty chunks

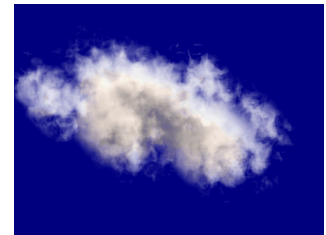
## Ray tracing – speeding up the calculations

- Bounding Volumes
  - Enclose groups of objects in sets of hierarchical bounding volumes (Octree)
  - First test for intersection with the bounding volume
  - Then only if there is an intersection, against the objects enclosed by the volume.



## Particle systems

- Used to simulate amorphous (shapeless) objects, e.g.
  - Fire,
  - Smoke
  - Water
  - Clouds



[http://cobweb.ecn.purdue.edu/~ebertd/635/Sp99/notes/Images/cloud\\_big.gif](http://cobweb.ecn.purdue.edu/~ebertd/635/Sp99/notes/Images/cloud_big.gif)

## Particle systems

- Amorphous objects are simulated by a discrete set of small particles
- Particles can have:
  - Mass
  - Position
  - Velocity
  - Temperature
  - Shape (a *sprite*)
  - Lifetime



[http://media.wiley.com/assets/430/48/0-07645-8789-7\\_1502.jpg](http://media.wiley.com/assets/430/48/0-07645-8789-7_1502.jpg)

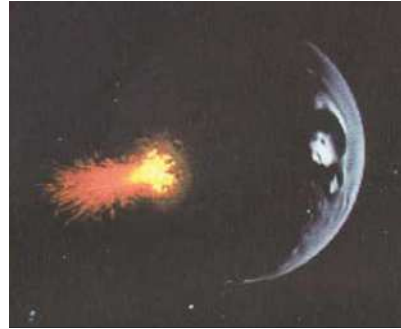
## Particle systems

- Particles can be created according to a probability distribution
- They can be given an initial velocity and a lifetime
- Depending on the simulation their movement can be determined by dynamics

## Particle system dynamics

- Newtonian particles:  
 $\mathbf{f} = m \mathbf{a}$  ( $\mathbf{f}$  and  $\mathbf{a}$  are vectors)  
 $\mathbf{a} = d\mathbf{v}/dt = d^2\mathbf{x}/dt^2$
- Given  $\mathbf{f}$  we need to find the change in position  $\mathbf{x}$
- Since we are working in frame intervals we can use a simple approximation  
 $\mathbf{v}_{t+\Delta t} = \mathbf{a}_t \Delta t + \mathbf{v}_t$   
 $\mathbf{x}_{t+\Delta t} = \mathbf{v}_t \Delta t + \mathbf{x}_t$

## Particle systems



## Credits

- This presentation has used slides from various web sources, including:
  - [http://lgm.fri.uni-lj.si/RG/KRIVLJENJE\\_3D/3D\\_krivljenje.ppt](http://lgm.fri.uni-lj.si/RG/KRIVLJENJE_3D/3D_krivljenje.ppt)
  - <http://www.doc.ic.ac.uk/~dfg/graphics/>
  - <http://www.utm.utoronto.ca/~arnold/320/04s/lectures/16/lecture-16.pdf>
  - [http://www.cs.drexel.edu/~david/Courses/CS431/Lectures/CGII\\_Pres7.pdf](http://www.cs.drexel.edu/~david/Courses/CS431/Lectures/CGII_Pres7.pdf)
  - <http://www.cs.unc.edu/~bloyd/comp770/Lecture12.pdf>
  - <http://hof.povray.org/>
  - <http://www.cs.otago.ac.nz/cosc342/2004-342/Textures2.pp4.pdf>
  - <http://www.avl.iu.edu/~ewernert/b581/lectures/15.1/bumpmap1.jpg>
  - [http://cobweb.ecn.purdue.edu/~ebertd/635/Sp99/notes/Images/cloud\\_big.gif](http://cobweb.ecn.purdue.edu/~ebertd/635/Sp99/notes/Images/cloud_big.gif)

