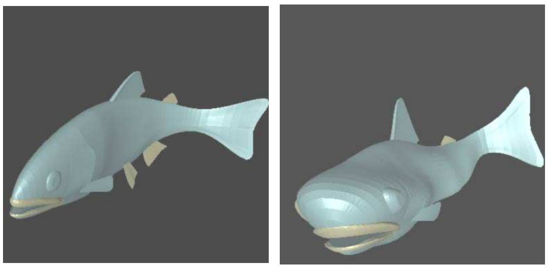
Adding realism

Overview of further methods

Warping and free-form deformation
Ray tracing
Particle systems

Warping and free-form deformation

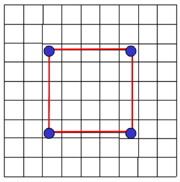


http://lgm.fri.uni-lj.si/RG/KRIVLJENJE_3D/3D_krivljenje.ppt

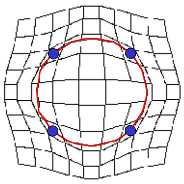
Warping and free-form deformation

- 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



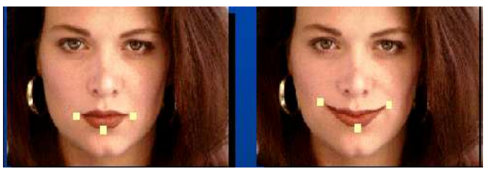
Define a mesh around an object



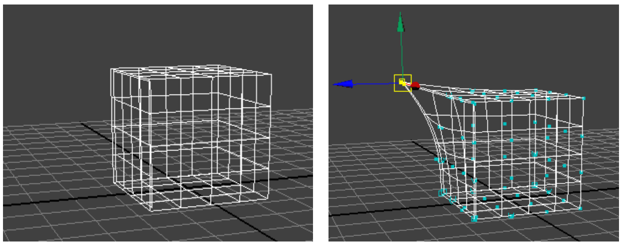
Deform the mesh at all the nodes as required.

Then interpolate the deformation for all the object vertices and pixels

Warping and free-form deformation 2D



Warping and free-form deformation 3D

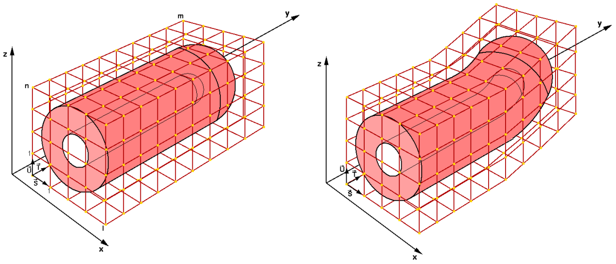


Define a mesh around an object

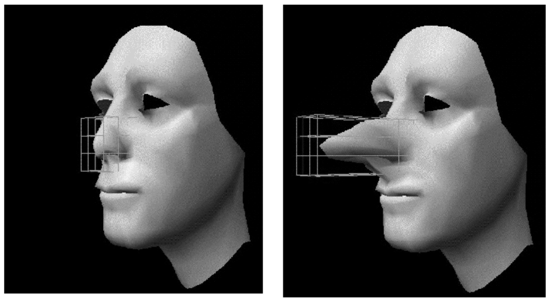
Deform the mesh at all the nodes as required.

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

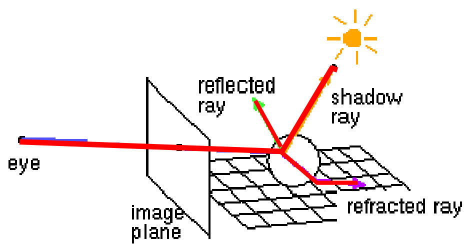
- Ray tracing is one of the most popular methods used in 3D computer graphics to render an image
- It works by tracing the path taken by a ray of light through the scene, and calculating reflection, refraction, or absorption of the ray whenever it intersects an object in the world
- Adds more realism 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 place
- Each ray is cast through the viewing volume and checked for intersections with the objects inside the viewing volume

Ray tracing



Ray tracing

- In practice a ray of light is traced in a backwards direction
- We start from the eye or camera and trace the ray through a pixel in the image plane into the scene and determine what it hits
- The pixel is then set to the colour values returned by the ray
- Each ray that hits an object can spawn other rays (reflected ray & refracted ray)

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

Backward ray tracing (cont)

- If the object is a light source, this is straightforward.
- Otherwise:
  - Recursively trace a new backward ray reflected by the surface.
  - Recursively trace a new backward ray from the intersection point to each light source on the "strike side" of the surface. That light source will contribute to the illumination of the intersection point.
  - The amount of light reaching the pixel depends on the angle of the ray to the surface, as well as absorption and refraction properties of the surface.
    - The light colour will be affected by the surface colour.

Ray tracing

Primary rays

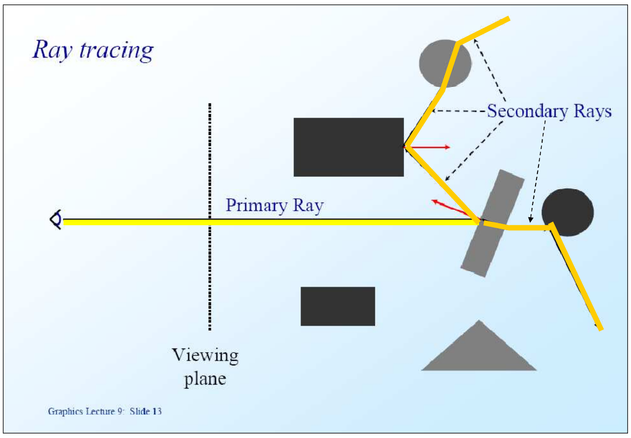
- Primary rays are rays from the viewpoint to the nearest intersection point
- For each ray we need to test which objects are intersecting the ray:
  - If the object has an intersection with the ray we calculate the distance between viewpoint and intersection
  - If the ray has more than one intersection, the smallest distance identifies the visible surface
- We compute the colour as discussed in an earlier lecture:

$$I = K_a I_a + K_d I_d \cos \theta_d + K_s I_s (\cos \theta_s)^n$$

Ray tracing

Secondary rays

- Secondary rays are rays originating at the intersection points
- Secondary rays are caused by
  - rays reflected off the intersection point in the direction of reflection
  - rays transmitted through transparent materials in the direction of refraction



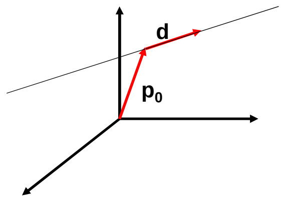
Ray tracing - calculations

- 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

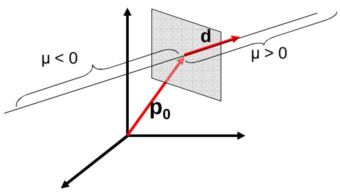
- Rays are parametric lines
- Can be defined by
  - origin $\mathbf{p_0}$
  - direction $\mathbf{d}$
- Equation of ray:
- $\mathbf{p} = \mathbf{p_0} + \mu\mathbf{d}$



Ray tracing - calculations

Calculating Intersection

- The viewing ray can be parameterized by μ:
  - $\mu > 0$ 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

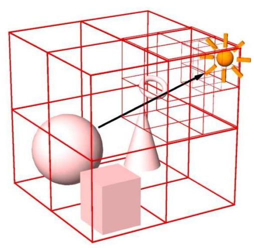
- We now have to calculate the point of intersection of each ray with each surface on its path
- Different surface representations require different solutions. See the website below for equations for sphere, cylinder, plane and triangle
- http://www.doc.ic.ac.uk/~dfg/graphics/GraphicsSlides11.pdf
- http://www.doc.ic.ac.uk/~dfg/graphics/GraphicsSlides12.pdf

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

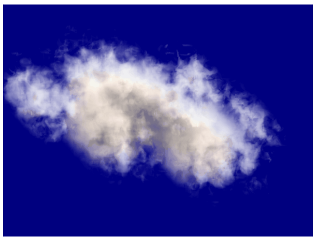
Ray tracing – speeding up the calculations

- 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

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

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

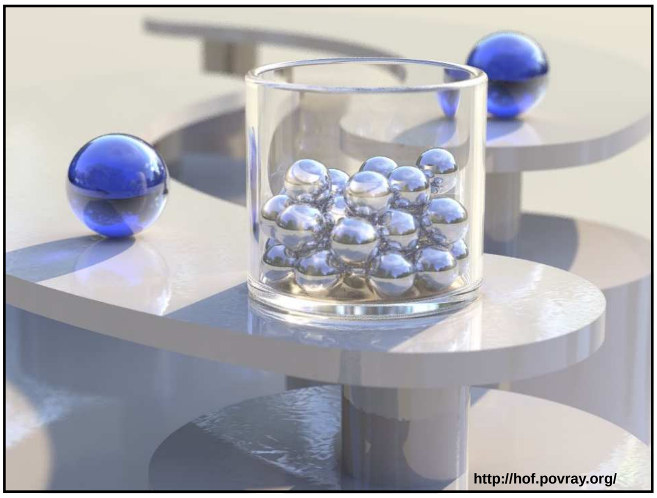
Particle system dynamics

- 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
  $\mathbf{v}_{t+1} = \mathbf{a}_t\,\Delta t + \mathbf{v}_t$
  $\mathbf{x}_{t+1} = \mathbf{v}_t\,\Delta t + \mathbf{x}_t$

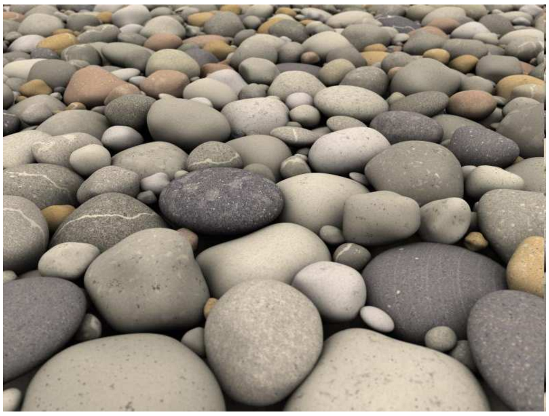
Particle systems

Credits

- This presentation has used slides from various web sources, including:
  - http://lgm.fri.uni-lj.si/RG/KRIVLJENJE_3D/3D_krivljenje.ppt
  - http://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/Classes/CS431/Lectures/CGII_Pres7.pdf
  - http://www.cs.unc.edu/~blloyd/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://hof.povray.org/



http://hof.povray.org/

http://hof.povray.org/