Five Key Problems in Computer Graphics

Penny Rheingans
UMBC

Computer Graphics

- Using computer to generate simulated scenes or worlds
- Requires tricking eye to believe 2D collection of pixels is really a continuous 3D world
- Coding-intensive application with strong basis in creativity and human perception

Five Key Problems

- What do you see?
- What does it look like?
- What shape is it?
- How does it move?
- Why does it have to look like a photograph?

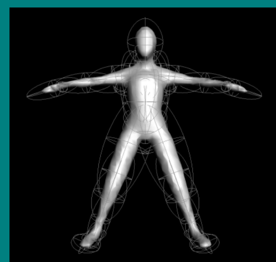
What shape is it?

Modeling Approaches

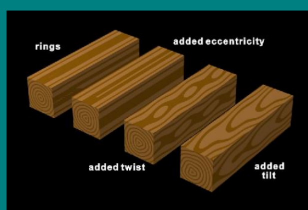
- Modeling problem
  - Define shape, color, and other visual properties
- Modeling solutions
  - Manual primitive creation
  - Scans from physical object
  - Functional descriptions
  - Grammar-based generation
  - Biologically-inspired simulations

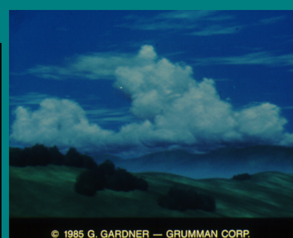
Scanning

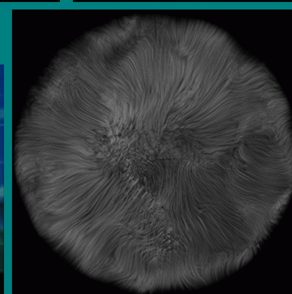
Functional Descriptions



- Define visual attributes with function, defined over space
  - Shape
  - Density
  - Color







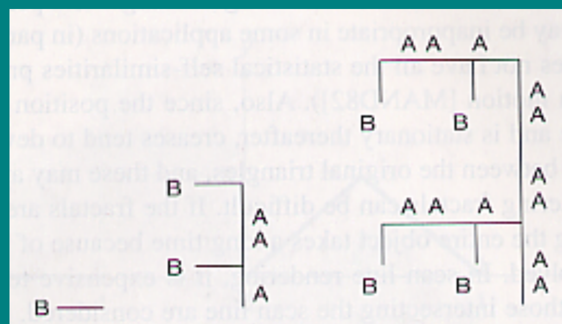
Grammar-based Generation

- Use (mostly) context-free grammars (CFG) to specify structural change over generations
- A CFG G=(V,T,S,P) where
  - V is a set of non-terminals
  - T is a set of terminals
  - S is the start symbol
  - P is a set of productions (rules) of the form:
    - $A \rightarrow x$, where $A \in V$, $x \in (V \cup T)^*$

Applying Grammar Rules

Rules
- B → A[B]AA[B]
- A → AA

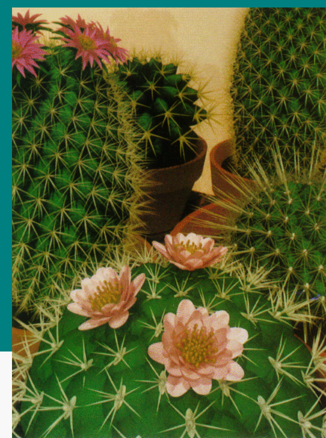
- Branches to left

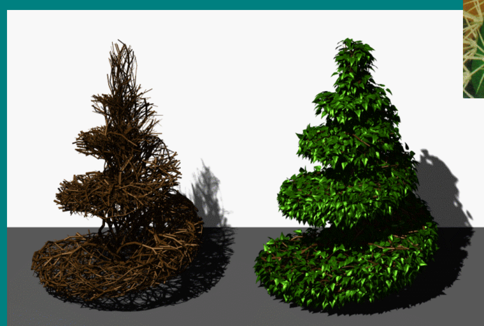


Strings
1: B
2: A[B]AA[B]
3: AA[A[BAA[B]]AAAA[A[B]AA[B]]

Applying Grammar Rules

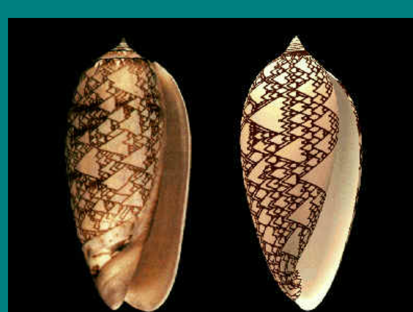
- N = 7, a = 25.7°
- S = X
- Rules:
  X→ F[+X][-X]FX
  F → FF

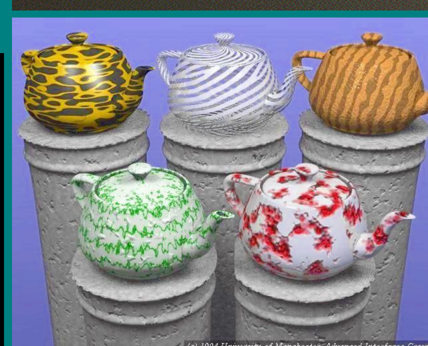




Biological Simulations

Mimic developmental process:
- cellular automata
- reaction diffusion

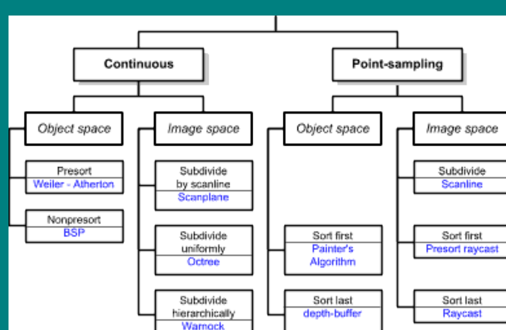




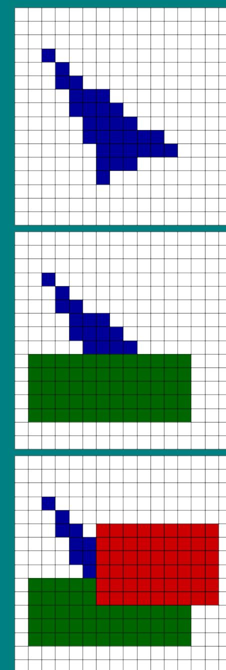
What do you see?

Visibility Approaches

- Visibility problem
  - Determine which objects (or parts of objects) are closest and therefore visible (a sorting problem)
- (Some) visibility solutions
  - Painter’s algorithm
  - Zbuffer
  - Scanline
  - Ray tracing



Painter's Algorithm



- Basic approach
  - Draw polygons, from farthest to closest
- First polygon:
  - (6,3,10), (11, 5,10), (2,2,10)
- Second polygon:
  - (1,2,8), (12,2,8), (12,6,8), (1,6,8)
- Third polygon:
  - (6,5,5), (14,5,5), (14,10,5),(6,10,5)

Painter's Algorithm

- Given

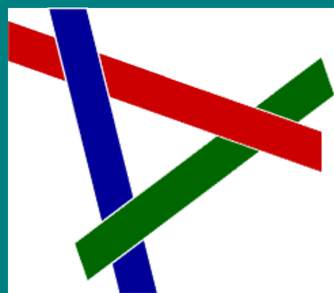
```
List of polygons {P1, P2, …. Pn)
An array of Intensity [x,y]
```

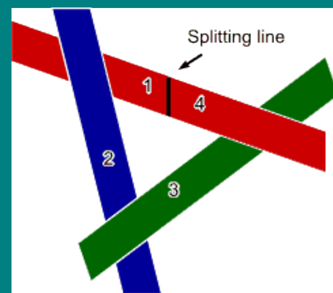
- Begin

```
Sort polygon list on minimum Z (largest z
  value comes first in sorted list)
For each polygon P in selected list do
   For each pixel (x,y) that intersects P do
      Intensity[x,y] = intensity of P at (x,y)
Display Intensity array
```

Painter's Algorithm: Cycles

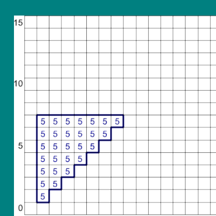
- Which to scan first?



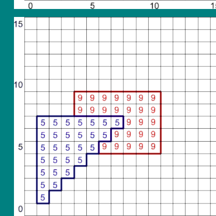


- Split along line, then scan 1,2,3,4 (or split another polygon and scan accordingly)
- Moral: Painter's algorithm is fast and easy, except for detecting and splitting cycles and other ambiguities

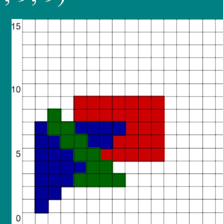
Z-Buffer

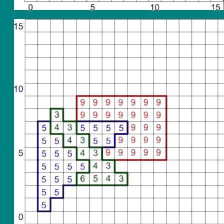


- Basic approach
  - Draw polygons, remembering depth of stuff drawn so far



- First polygon
  (1, 1, 5), (7, 7, 5), (1, 7, 5)
- Second polygon
  (3, 5, 9), (10, 5, 9), (10, 9, 9), (3, 9, 9)
- Third polygon
  (2, 6, 3), (2, 3, 8), (7, 3, 3)





Z-Buffer Algorithm

- Given

  List of polygons {P_1, P_2, …., P_n}

  An array x-buffer[x,y] initialized to +infinity

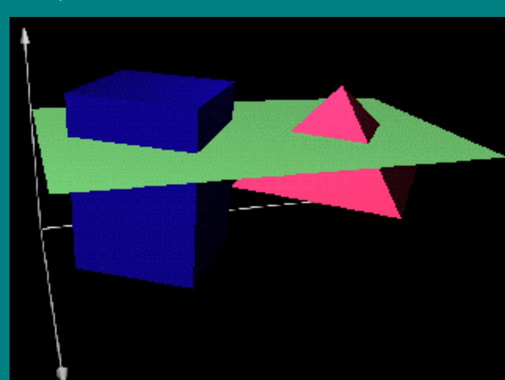
  An array Intensity[x,y]

- Begin

```
For each polygon P in selected list do
   For each pixel (x,y) that intersects P do
      Calculate z-depth of P at (x,y)
      If z-depth < z-buffer[x,y] then
         Intensity[x,y] = intensity of P at (x,y)
         Z-buffer[x,y] = z-depth
Display Intensity array
```

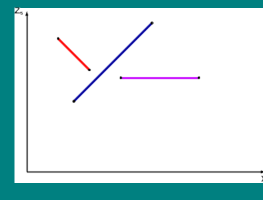
Scanline Algorithm

- Basic approach
  - Simply problem by considering only one scanline at a time (3D problem -> 2D)

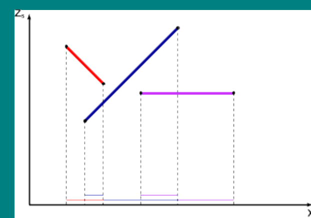


Scanline Algorithm

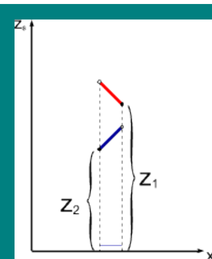
- Consider xz slice



- Calculate where visibility can change



- Decide visibility in each span

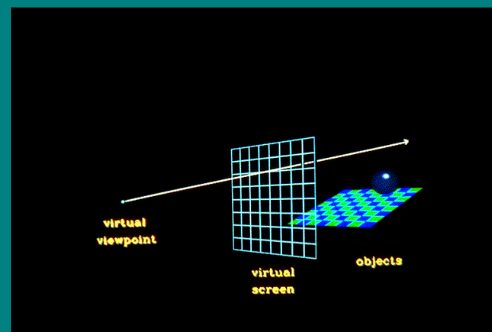


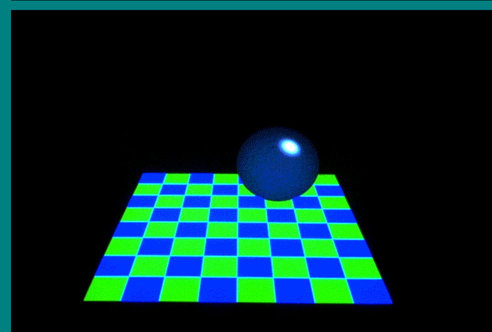
Scanline Algorithm

```
1.  Sort pgons into sorted surface table (SST) on Y
2.  Initialize y and active surface table (AST)
    Y = first nonempty scanline
    AST = SST[y]
3.  Repeat until AST and SST are empty
    Identify spans for this scanline (sorted on x)
    For each span
       determine visible element (based on z)
       fill pixel intensities with values from pgon
    Update AST
       remove exhausted polygons
       y++
       update x intercepts
       resort AST on x
       add entering polygons
4.  Display Intensity array
```

Raytracing

- Basic approach
  - Cast ray from viewpoint through pixels into scene





Raytracing Algorithm

```
Given
    List of polygons { P1, P2, ..., Pn }
    An array of intensity [ x, y ]
{
   For each pixel (x, y) {
      form a ray R in object space through the
        camera position C and the pixel (x, y)
      Intensity [ x, y ] = trace ( R )
      }
   Display array Intensity
}
```

Raytracing Algorithm

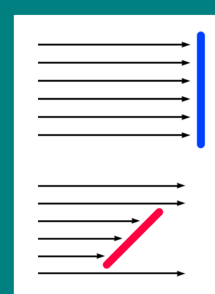
```
intensity Function  trace ( Ray )
{
   for each polygon P in the scene
      calculate the intersection of P and the ray R
   if ( The ray R hits no polygon )
        return ( background intensity )
   else {
      find the polygon P with the closest
        intersection
      calculate intensity I at intersection point
      return ( Illuminate( I, trace(reflect ),
        trace( refract ) ) )
    }
}
```

What does it look like?

Illumination Approaches

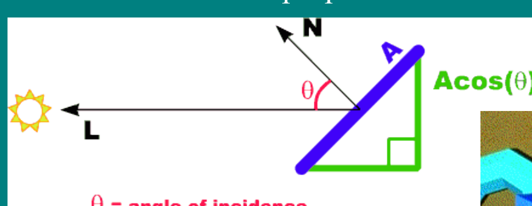
- Illumination problem
  - Model how objects interact with light
- Modeling solutions
  - Simple physics/optics
  - More realistic physics
    - Surface physics
    - Surface microstructure
    - Subsurface scattering
    - Shadows
    - Light transport

Simple Optics: Diffuse Reflection



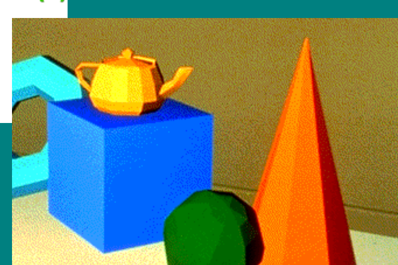
Lambert's Law:

the radiant energy from any small surface area dA in any direction θ relative to the surface normal is proportional to cos θ



$I_{diff} = k_d I_l \cos\theta$

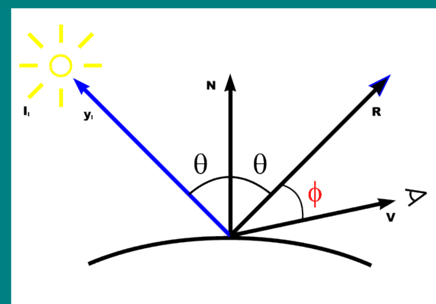
$= k_d I_l \,(N \bullet L)$



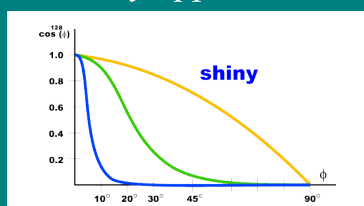
Simple Optics: Specular Reflection

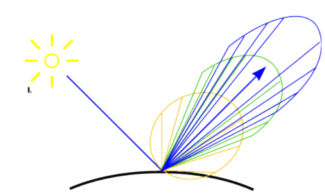
For specific wavelength λ

$$I_{spec\lambda} = k_{s\lambda} I_{\lambda} \cos^{n}\phi$$
$$= k_{s\lambda} I_{\lambda} (R \bullet V)^{n}$$



Hacky approximation for shinyness



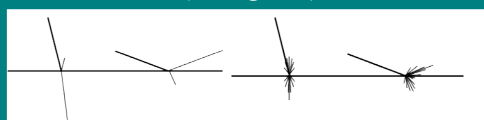


Simple Optics: Refraction

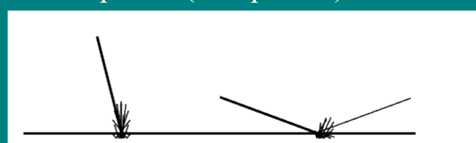
Surface Physics

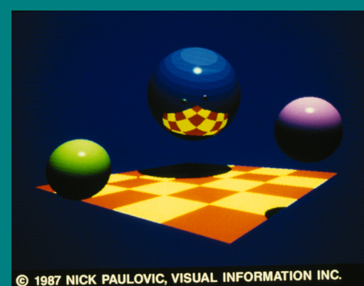
- Conductor (like metal)

- Dielectric (like glass)

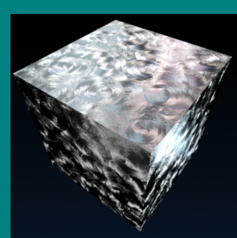


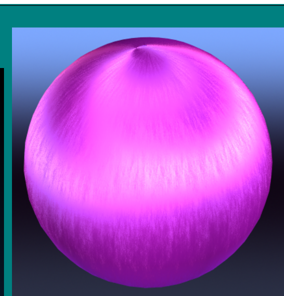
- Composite (like plastic)

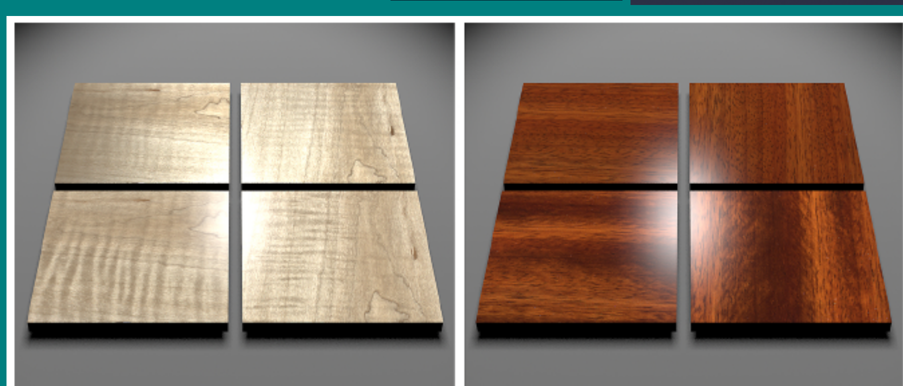




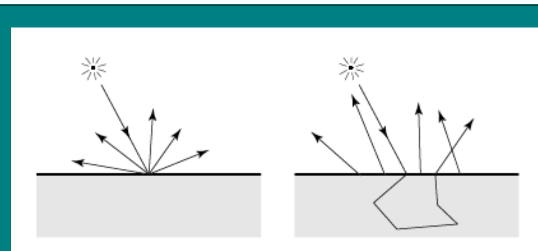
Surface Microstructure

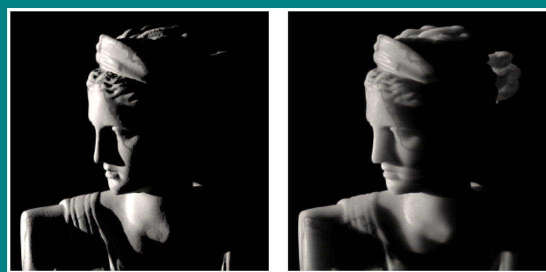




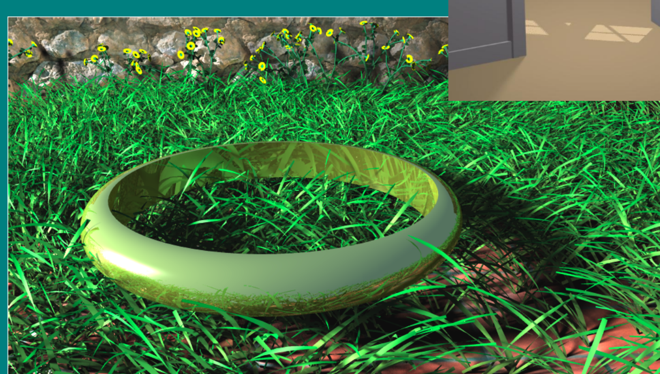


Subsurface Scattering

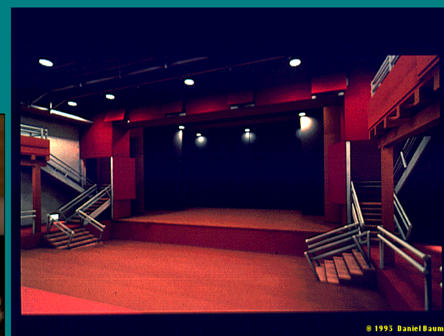




Shadows



Light Transport

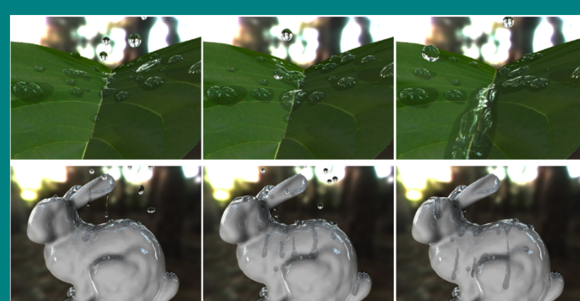


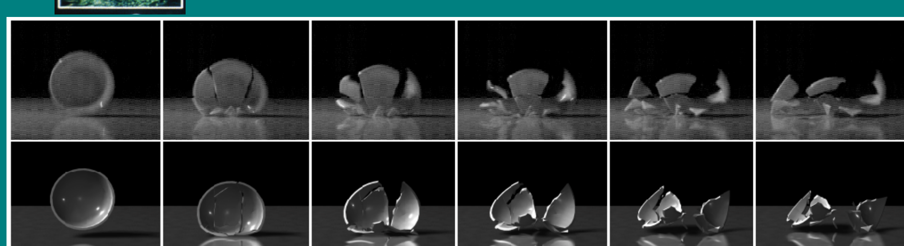
How does it move?

Motion Dynamics Approaches

- Motion dynamics problem
  - Define geometric movements and deformations of objections under motion
- Dynamics solutions
  - Simulate physics of simple objects
  - Model structure and constraints
  - Capture motion from reality
  - Simulate group dynamics
  - Use your imagination

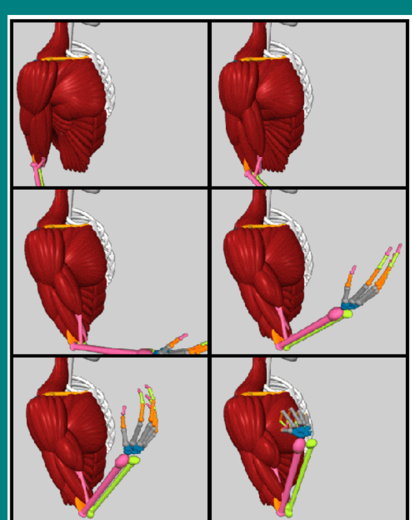
Simulate Physics

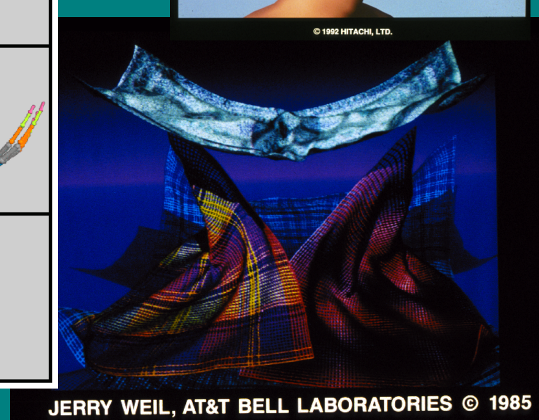




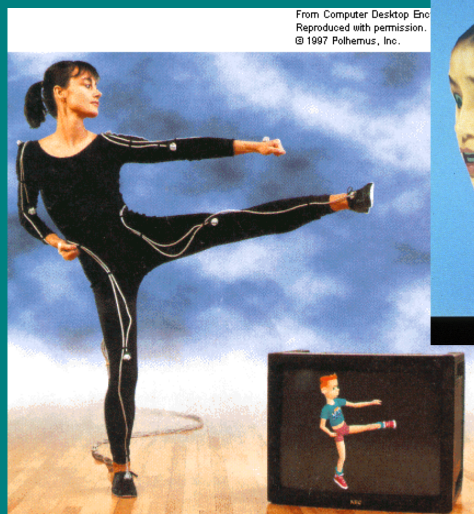
Graphics for Training

Model Structure

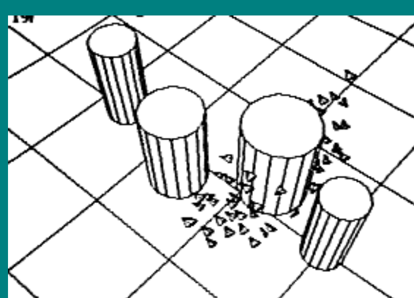


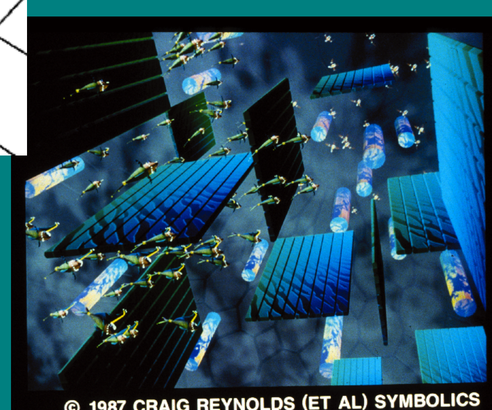


Motion Capture



Behavioral Simulation





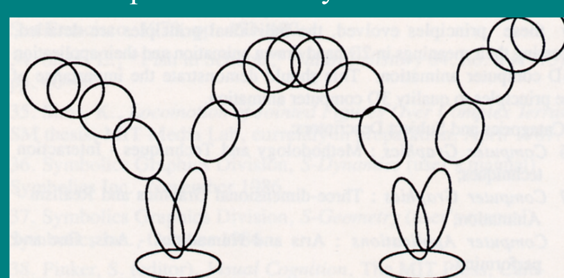
Use Your Imagination

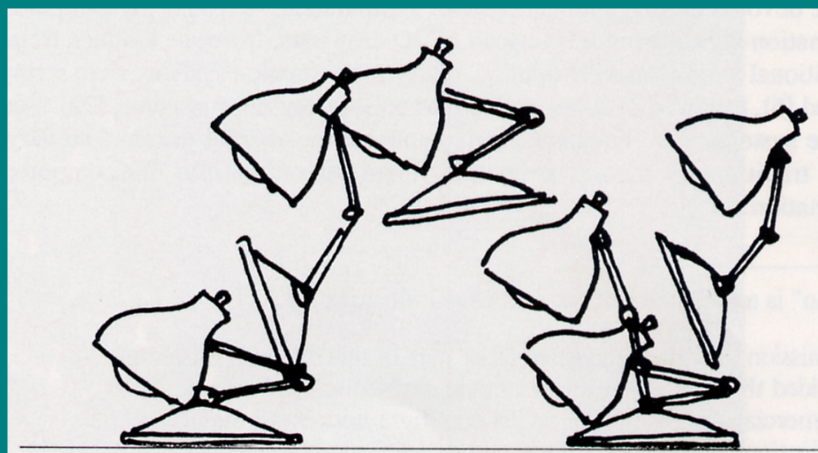
John Lasseter

Play

Squash and Stretch

- Defining the rigidity and mass of an object by distorting its shape during an action
- Keys
  - Volume constant
  - Different materials respond differently



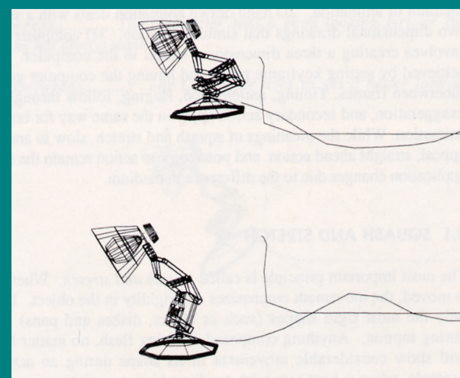


Anticipation

- The preparation for an action
- Ex:
  - Pull back foot to kick ball
  - Luxo: big lamp looks off stage before Jr.'s entrance
- Keys
  - Direct attention to upcoming action
  - Anticipation can allow faster action

Secondary Action

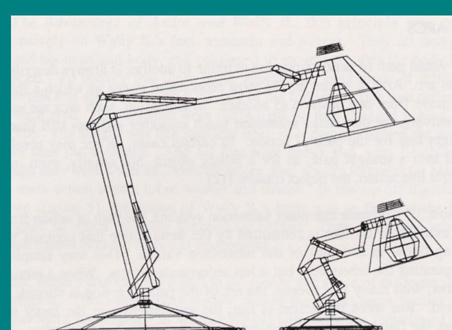
- Action that results directly from another action
- Ex:
  - Luxo: cord movement
  - Facial expression



- Keys
  - Needs to be subordinate to primary action

Appeal

- Design or action that the audience enjoys watching
- Ex:
  - Jr scaled like child



- Keys
  - Personality of characters (batting motions of two lamps)
  - Identify and express emotional state (Luxo hops)