Math Modeling Final Project

APPLICATIONS of FRACTALS

- FRACTAL LANDSCAPES
- FRACTAL IMAGE COMPRESSION

Advisor: Professor Alber

Fang Qi
Pu Wan
Xue Rui

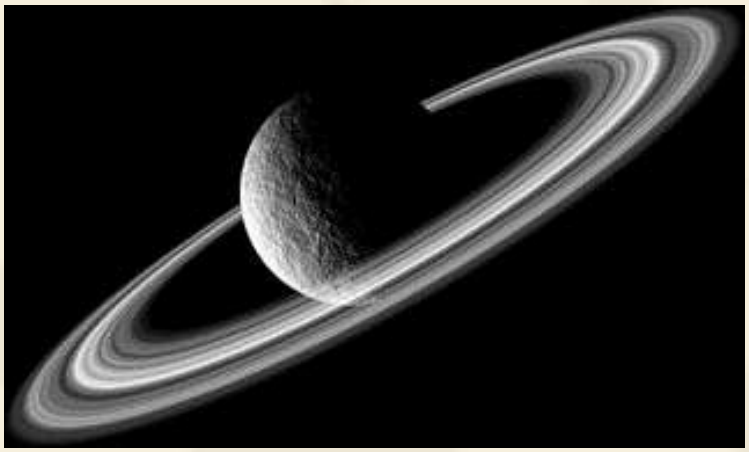
Review:

Two important properties of a fractal F

- F has detail at every level.
- F is exactly, approximately or statistically self-similar.

Fractal Landscapes

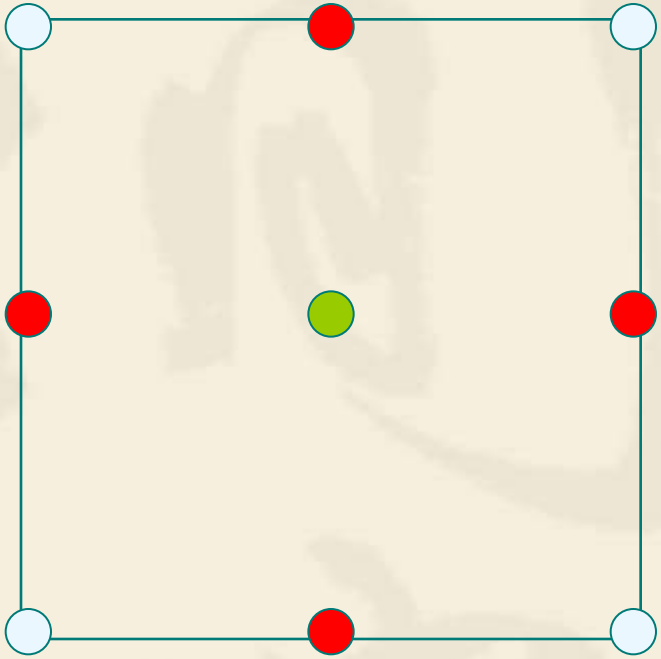
Fractals are now used in many forms to create textured landscapes and other intricate models. It is possible to create all sorts of realistic *fractal forgeries*, images of natural scenes, such as lunar landscapes, mountain ranges and coastlines. This is seen in many special effects within Hollywood movies and also in television advertisements.



A fractal planet.

A fractal landscape created by Professor Ken Musgrave (Copyright: Ken Musgrave)

Simulation process



First:

- Already known
- The average of the 2 neighbor blue points plus *r*
- The average of the 4 neighbor red points plus *r*

This random value *r* is normally distributed and scaled by a factor d_1 related to the original *d* by

$$r \sim N(0, d_1) \quad d_1 = (\frac{1}{2})^{H/2} d$$

d is the scale constant.
H is the smoothness constant.

We can now carry out exactly the same procedure on each of the four smaller squares, continuing for as long as we like, but where we make the scaling factor at each stage smaller and smaller; by doing this we ensure that as we look closer into the landscape, the 'bumps' in the surface will become smaller, just as for a real landscape. The scaling factor at stage *n* is *dn*, given by

$$d_n = (\frac{1}{2})^{nH/2} d$$

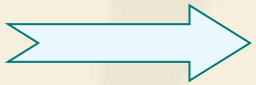
Simulation result by MATLAB

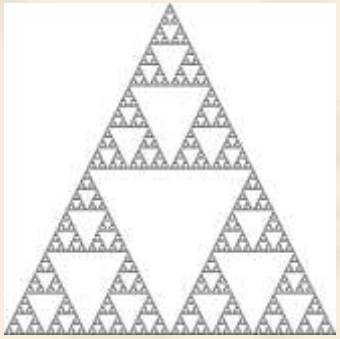
Using a 64x64 grid and

H=1.25
d=15

FRACTAL IMAGE COMPRESSION

What is Fractal Image Compression ?





The output images converge to the Sierpinski triangle. This final image is called *attractor* for this photocopying machine. Any initial image will be transformed to the attractor if we repeatedly run the machine.

On the other words, the attractor for this machine is always the same image without regardless of the initial image. This feature is one of the keys to the fractal image compression.

How can we describe behavior of the machine ? Transformations of the form as follows will help us.

$$w_i \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix}$$

Such transformations are called *affine transformations*. Affine transformations are able to skew, stretch, rotate, scale and translate an input image.

M.Barnsley suggested that perhaps storing images as collections of transformations could lead to image compression.

Iterated Function Systems (IFS)

An iterated function system consists of a collection of contractive affine transformations.

$$W(*) = \bigcup_{i=1}^{n} w_i(*)$$

For an input set S, we can compute wi for each i, take the union of these sets, and get a new set *W(S)*.

Hutchinson proved that in IFS, if the wi are contractive, then *W* is contractive, thus the map *W* will have a unique fixed point in the space of all images. That means, whatever image we start with, we can repeatedly apply *W* to it and our initial image will converge to a fixed image. Thus *W* completely determine a unique image.

$$|W| \equiv f_\infty = \lim_{x \to \infty} W^{(n)}(f_o)$$

Self-Similarity in Images

We define the distance of two images by:

$$\delta(f,g) = \sup_{(x,y)\in P} |f(x,y) - g(x,y)|$$

where *f* and *g* are value of the level of grey of pixel, *P* is the space of the image

Original Lena image

Self-similar portions of the image

JAVA

Affine transformation mentioned earlier is able to "geometrically" transform part of the image but is not able to transform grey level of the pixel. so we have to add a new dimension into affine transformation

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

Here s_i represents the contrast, o_i the brightness of the transformation

For encoding of the image, we divide it into:

- non-overlapped pieces (so called *ranges, R*)
- overlapped pieces (so called *domains, D*)

Encoding Images

Suppose we have an image *f* that we want to encode. On the other words, we want to find a IFS *W* which *f* to be the fixed point of the map *W*

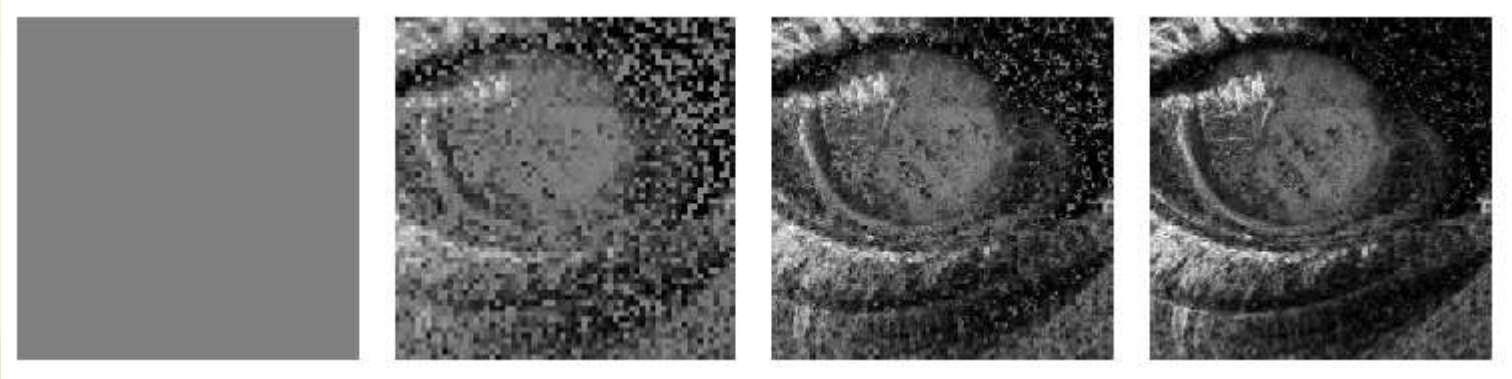
We seek a partition of f into N non-overlapped pieces of the image to which we apply the transforms *wi* and get back *f*.

We should find pieces Di and maps wi, so that when we apply a wi to the part of the image over Di , we should get something that is very close to the any other part of the image over Ri .

Finding the pieces Ri and corresponding Di by minimizing distances between them is the goal of the problem.

Decoding Images

The decoding step is very simple. We start with any image and apply the stored affine transformations repeatedly till the image no longer changes or changes very little. This is the decoded image.



First three iterations of the decompression of the image of an eye from an initial solid grey image

An Example

Suppose we have to encode 256x256 pixel grayscale image.

let R_1~R_{1024} be the 8x8 pixel non-overlapping sub-squares of the image, and let *D* be the collection of all overlapping 16x16 pixel sub-squares of the image (general, domain is 4 times greater than range). The collection *D* contains (256-16+1) * (256-16+1) = 58,081 squares. For each Ri search through all of *D* to find Di with minimal distances

To find most-likely sub-squares we have to minimize distance equation. That means we must find a good choice for Di that most looks like the image above Ri (in any of 8 ways of orientations) and find a good contrast si and brightness oi. For each Ri , Di pair we can compute contrast and brightness using least squares regression.

Fractal Image Compression versus JPEG Compression

Original Lena image
(184,320 bytes)

JPEG-max. quality
(32,072)
comp. ratio: **5.75:1**

FIF-max. quality
(30,368)
comp. ratio: **6.07:1**

Resolution Independence

One very important feature of the Fractal Image Compression is *Resolution Independence*.

When we want to decode image, the only thing we have to do is apply these transformations on any initial image. After each iteration, details on the decoded image are sharper and sharper. That means, the decoded image can be decoded at any size.

So we can zone in the image on the larger sizes without having the "pixelization" effect..

Lena's eye
original image enlarged to 4 times

Lena's eye
decoded at 4 times its encoding size

Thank You!

SOLID MODELLING

Why solid modeling?

- Recall weakness of wireframe and surface modeling
  - Ambiguous geometric description
  - incomplete geometric description
  - lack topological information
  - Tedious modeling process
  - Awkward user interface

Solid model

- Solid modeling is based on *complete, valid and unambiguous* geometric representation of physical object.
  - Complete → points in space can be classified.(inside/ outside)
  - Valid →vertices, edges, faces are connected properly.
  - Unambiguous → there can only be one interpretation of object

Solid model

- Analysis automation and integration is possible only with solid models→ has properties such as weight, moment of inertia, mass.
- Solid model consist of geometric and topological data
  - Geometry → shape, size, location of geometric elements
  - Topology →connectivity and associativity of geometric elements →non graphical, relational information

Solid model representation schemes

1. Constructive solid geometry (CSG)
2. Boundary representation (B-rep)
3. Spatial enumeration
4. Instantiation.

Constructive solid geometry (CSG)

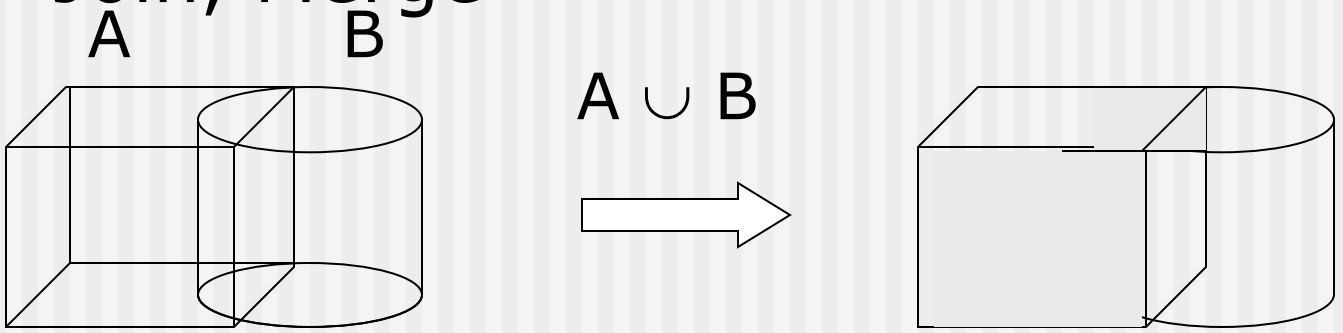
- Objects are represented as a combination of simpler solid objects (*primitives*).
- The primitives are such as cube, cylinder, cone, torus, sphere etc.
- Copies or “instances” of these primitive shapes are created and positioned.
- A complete solid model is constructed by combining these “instances” using set specific, logic operations (Boolean)

Constructive solid geometry (CSG)

- Boolean operation
  - each primitive solid is assumed to be a set of points, a boolean operation is performed on point sets and the result is a solid model.
  - Boolean operation → union, intersection and difference
  - The relative location and orientation of the two primitives have to be defined before the boolean operation can be performed.
  - Boolean operation can be applied to two solids other than the primitives.

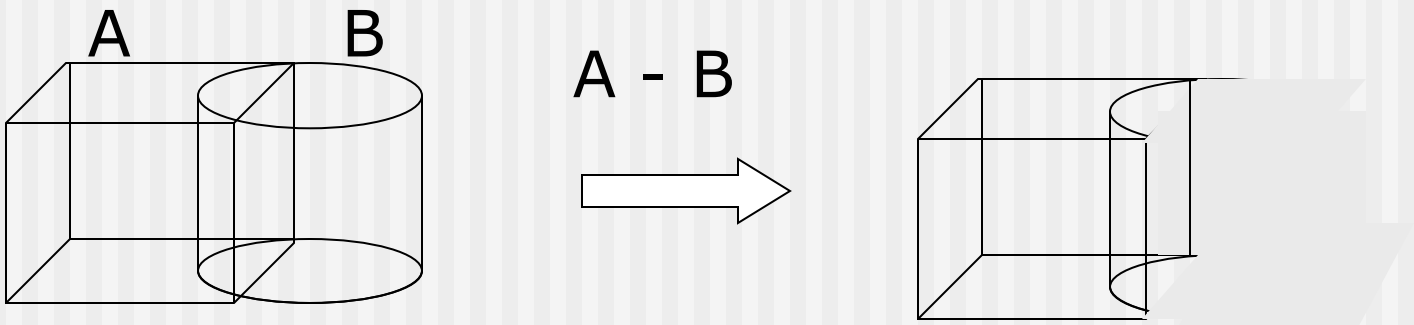
Constructive solid geometry (CSG)- boolean operation

- Union
  - The sum of all points in each of two defined sets. (logical “OR”)
  - Also referred to as Add, Combine, Join, Merge



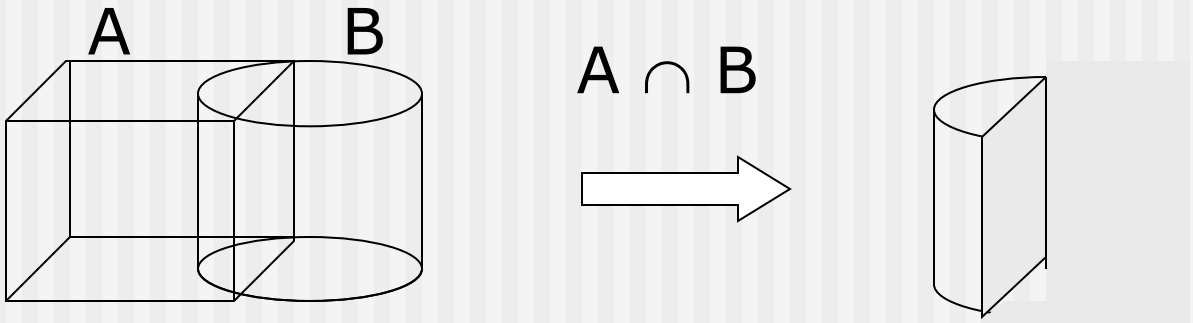
Constructive solid geometry (CSG)- boolean operation

- Difference
  - The points in a source set minus the points common to a second set. (logical “NOT”)
  - Set must share common volume
  - Also referred to as subtraction, remove, cut



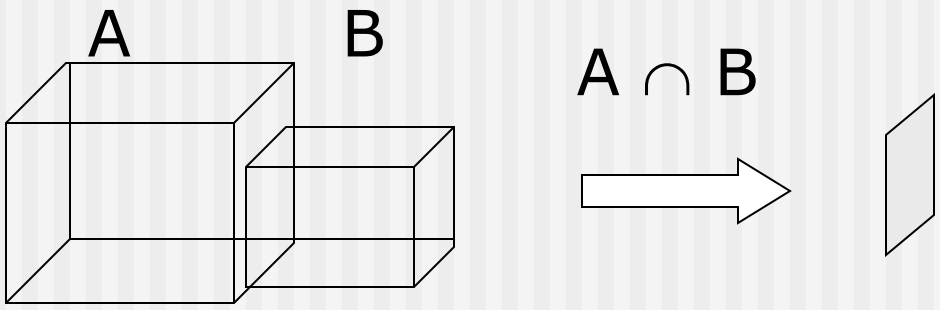
Constructive solid geometry (CSG)- boolean operation

- intersection
  - Those points common to each of two defined sets (logical “AND”)
  - Set must share common volume
  - Also referred to as common, conjoin



Constructive solid geometry (CSG)- boolean operation

- When using boolean operation, be careful to avoid situation that do not result in a valid solid



Constructive solid geometry (CSG)- boolean operation

- Boolean operation
  - Are intuitive to user
  - Are easy to use and understand
  - Provide for the rapid manipulation of large amounts of data.
- Because of this, many non-CSG systems also use Boolean operations

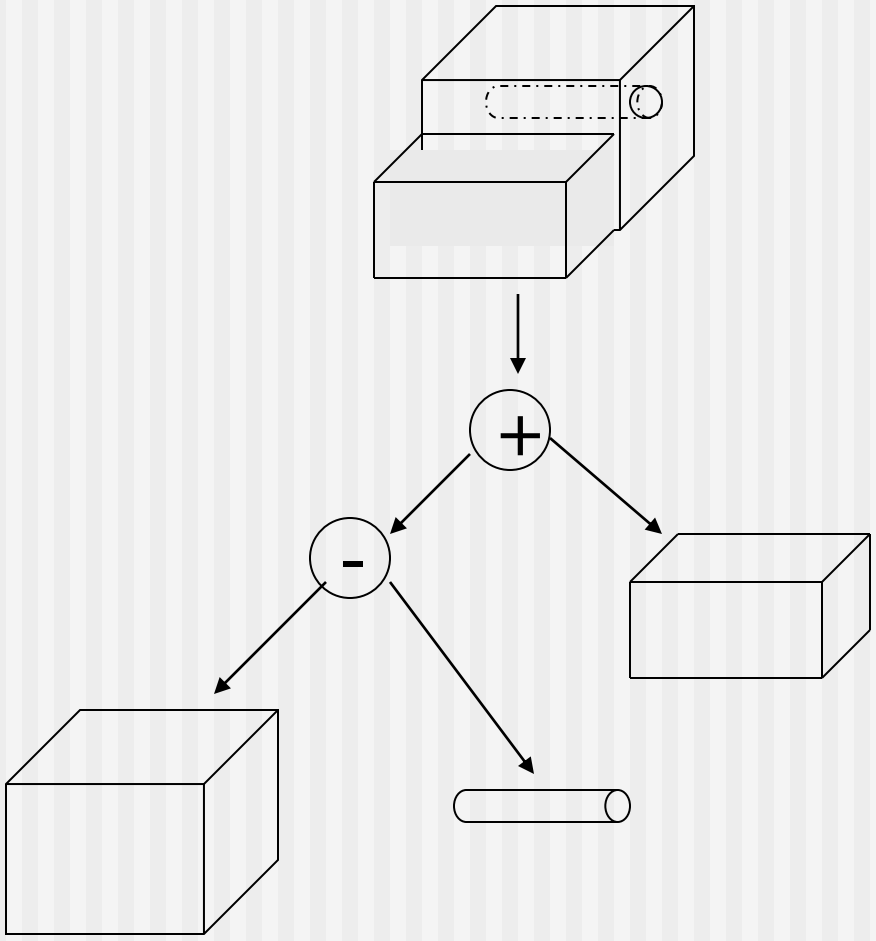
Constructive solid geometry (CSG)- data structure

- Data structure does not define model shape explicitly but rather implies the geometric shape through a procedural description
  - E.g: object is not defined as a set of edges & faces but by the instruction : *union primitive1 with primitive 2*
- This procedural data is stored in a data structure referred to as a CSG tree
- The data structure is simple and stores compact data → easy to manage

Constructive solid geometry (CSG)- CSG tree

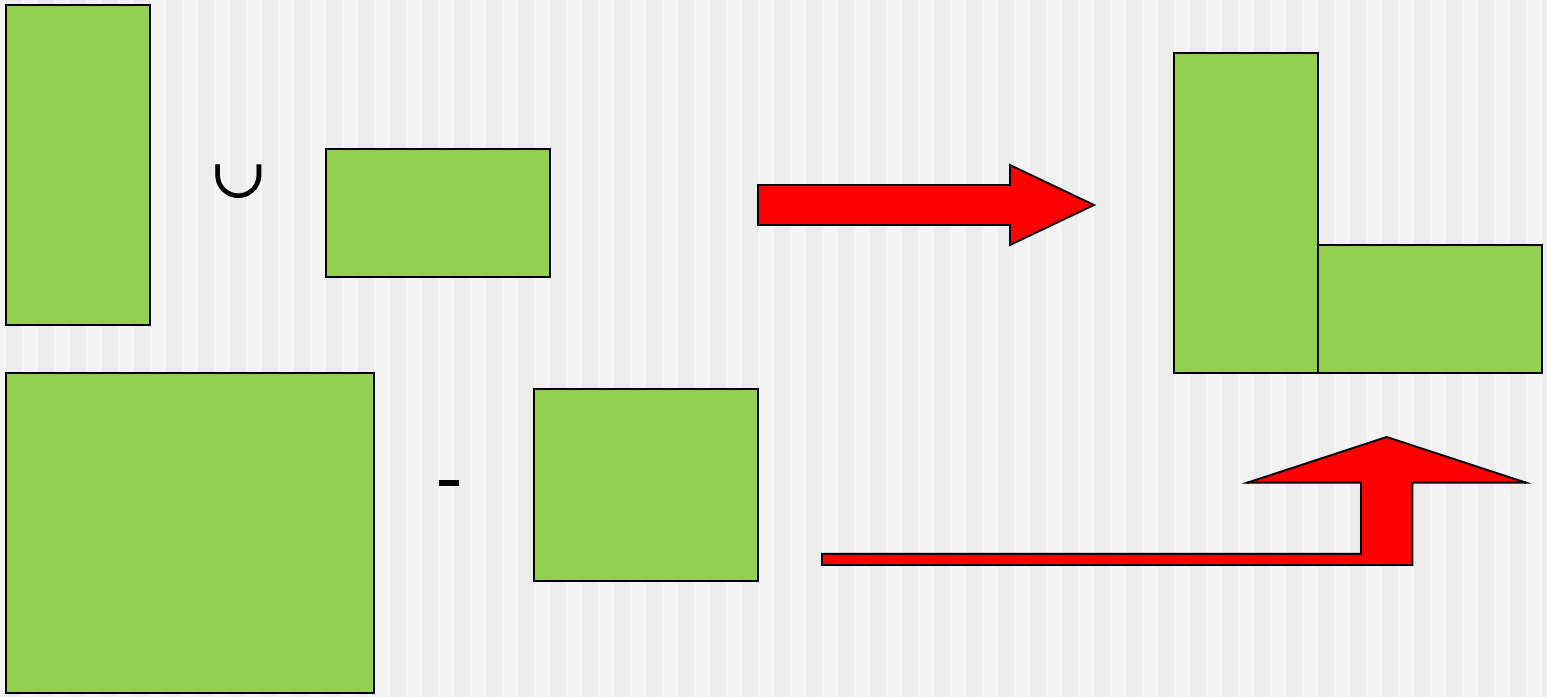
- CSG tree → stores the history of applying boolean operations on the primitives.
  - Stores in a binary tree format
  - The outer leaf nodes of tree represent the primitives
  - The interior nodes represent the boolean operations performed.

Constructive solid geometry (CSG)- CSG tree



Constructive solid geometry (CSG)- not unique

- More than one procedure (and hence database) can be used to arrive at the same geometry.



Constructive solid geometry (CSG) representation

- CSG representation is unevaluated
  - Faces, edges, vertices not defined in explicit
- CSG model are always valid
  - Since built from solid elements.
- CSG models are complete and unambiguous

Constructive solid geometry (CSG) - advantage

- CSG is powerful with high level command.
- Easy to construct a solid model – minimum step.
- CSG modeling techniques lead to a concise database→ less storage.
  - Complete history of model is retained and can be altered at any point.
- Can be converted to the corresponding boundary representation.

Constructive solid geometry (CSG) - disadvantage

- Only boolean operations are allowed in the modeling process → with boolean operation alone, the range of shapes to be modeled is severely restricted → not possible to construct unusual shape.
- Requires a great deal of computation to derive the information on the boundary, faces and edges which is important for the interactive display/ manipulation of solid.

solution

- CSG representation tends to accompany the corresponding boundary representation → *hybrid representation*
- Maintaining consistency between the two representations is very important.

Boundary representation (B-Rep)

- Solid model is defined by their enclosing surfaces or boundaries. This technique consists of the geometric information about the faces, edges and vertices of an object with the topological data on how these are connected.

Boundary representation (B-Rep)

- Why B-Rep includes such topological information?
  - A solid is represented as a closed space in 3D space (surface connect without gaps)
  - The boundary of a solid separates points inside from points outside solid.

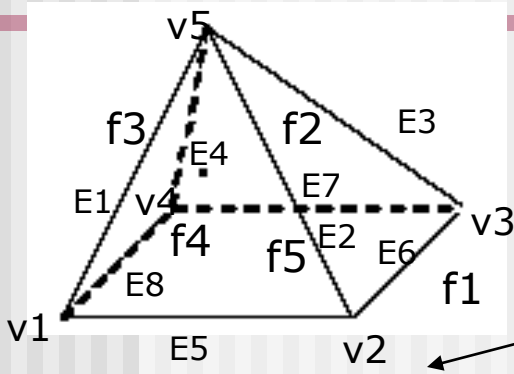
B-Rep vs surface modeling

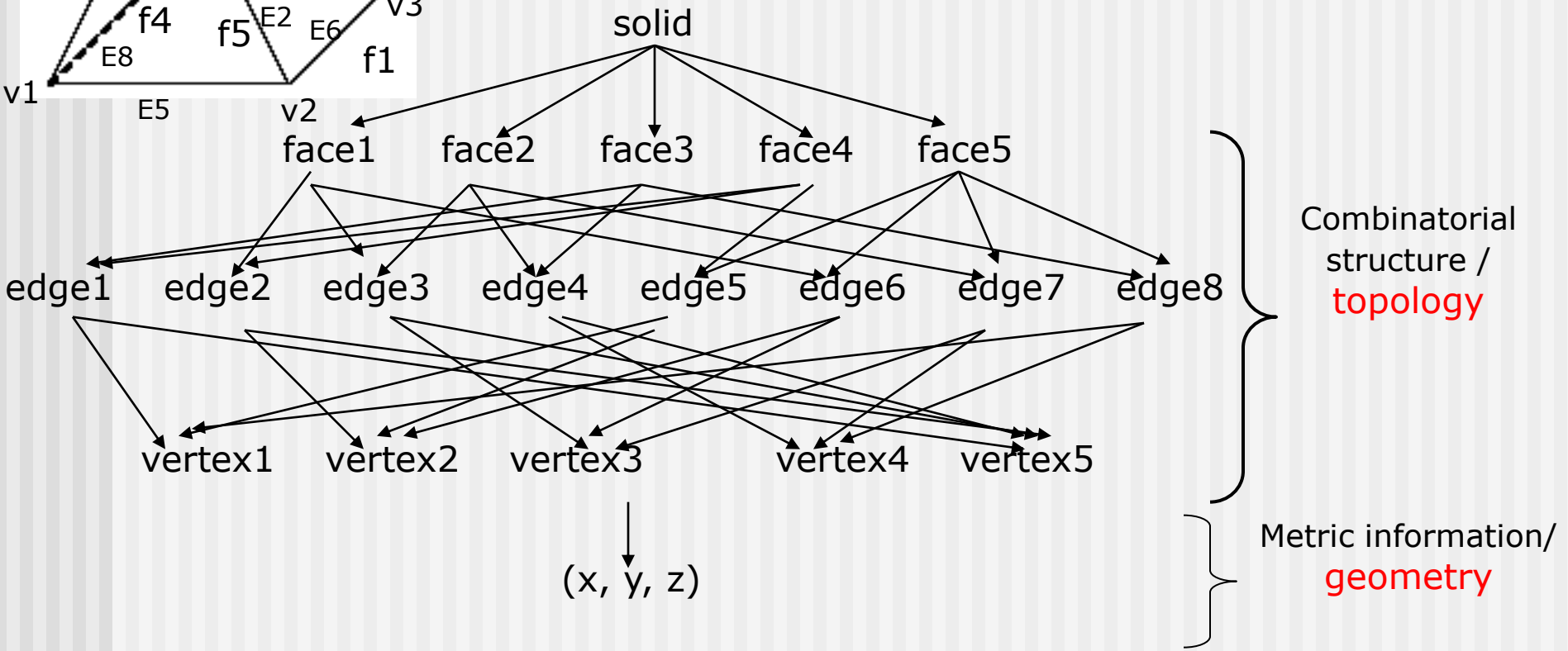
- Surface model
  - A collection of surface entities which simply enclose a volume lacks the connective data to define a solid (i.e topology).
- B- Rep model
  - Technique guarantees that surfaces definitively divide model space into solid and void, even after model modification commands.

B-Rep data structure

- B-Rep graph store face, edge and vertices as nodes, with pointers, or branches between the nodes to indicate connectivity.

B-Rep data structure





Boundary representation- validity

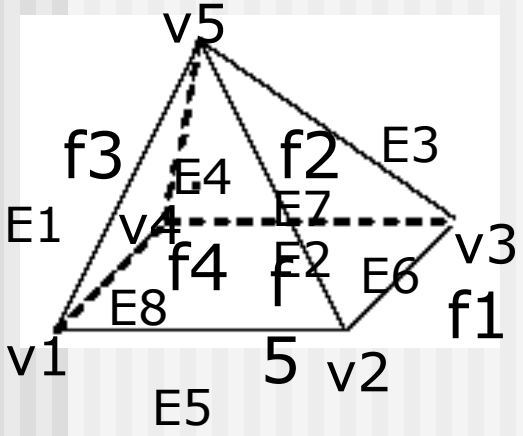
- System must validate topology of created solid.
- B-Rep has to fulfill certain conditions to disallow self-intersecting and open objects
- This condition include
  - Each edge should adjoin exactly two faces and have a vertex at each end.
  - Vertices are geometrically described by point coordinates

Boundary representation- validity

- This condition include (cont)
  - At least three edges must meet at each vertex.
  - Faces are described by surface equations
  - The set of faces forms a complete skin of the solid with no missing parts.
  - Each face is bordered by an ordered set of edges forming a closed loop.
  - Faces must only intersect at common edges or vertices.
  - The boundaries of faces do not intersect themselves

Boundary representation- validity

- Validity also checked through mathematical evaluation
  - Evaluation is based upon Euler's Law (valid for simple polyhedra – no hole)
  - V – E + F = 2 V-vertices E- edges F- face loops

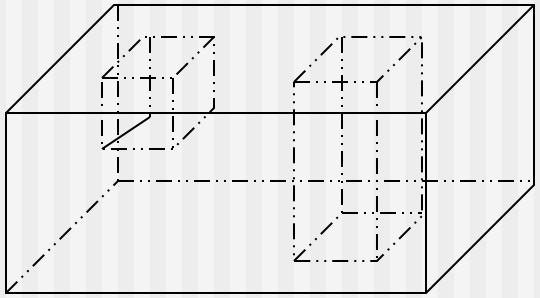


V = 5, E = 8, F = 5

5 – 8 + 5 = 2

Boundary representation- validity

- Expanded Euler's law for complex polyhedrons (with holes)
- Euler-Poincare Law:
  - V-E+F-H=2(B-P)
  - H – number of holes in face, P- number of passages or through holes, B- number of separate bodies.



V = 24, E=36, F=15, H=3, P=1,B=1

Boundary representation-ambiguity and uniqueness

- Valid B-Reps are unambiguos
- Not fully unique, but much more so than CSG
- Potential difference exists in division of
  - Surfaces into faces.
  - Curves into edges

Boundary representation-advantages

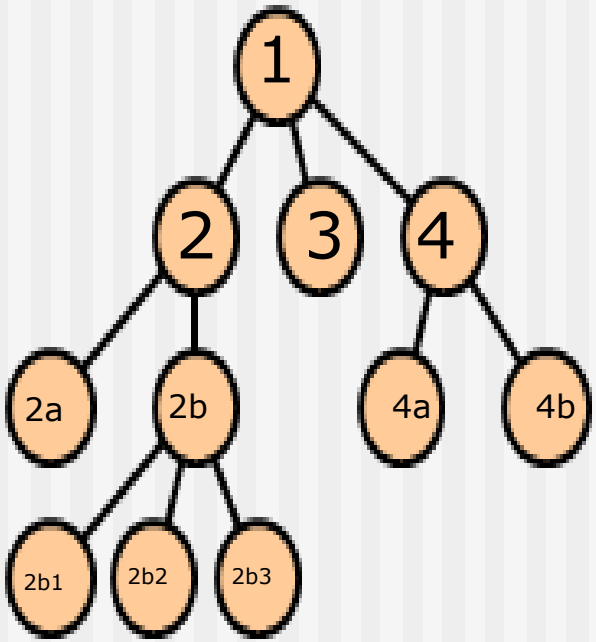
- Capability to construct unusual shapes that would not be possible with the available CSG→ aircraft fuselages, swing shapes
- Less computational time to reconstruct the image

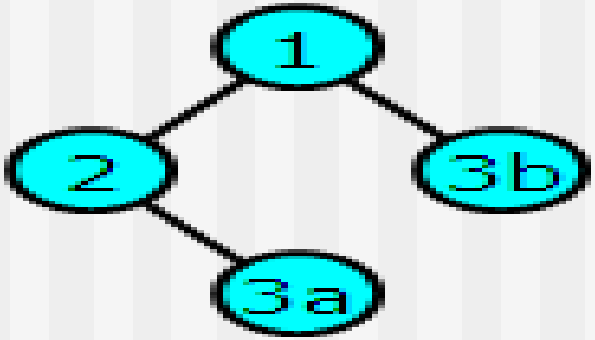
Boundary representation- disadvantages

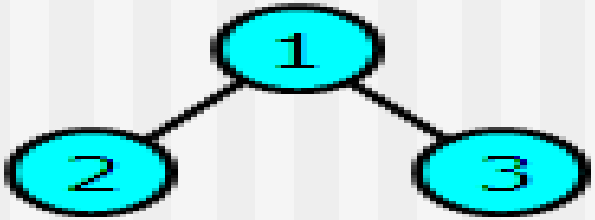
- Requires more storage
- More prone to validity failure than CSG
- Model display limited to planar faces and linear edges
  - complex curve and surfaces only approximated

BSP Trees, Quadtrees & Octrees

BSP Trees







Quadtrees & Octrees

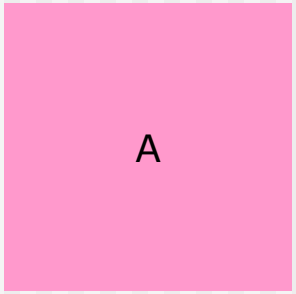
- Quadtrees are used to partition 2-D space, while octrees are for 3-D.

Quadtrees & Octrees: Definition

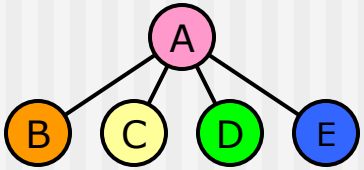
- In general:
  - A *quadtree* is a tree in which each node has at most 4 children.
  - An *octree* is a tree in which each node has at most 8 children.
  - Similarly, a *binary tree* is a tree in which each node has at most 2 children.
- In practice, however, we use "quadtree" and "octree" to mean something more specific:
  - Each node of the tree corresponds to a square (quadtree) or cubical (octree) region.
  - If a node has children, think of its region being chopped into 4 (quadtree) or 8 (octree) equal subregions. Child nodes correspond to these smaller subregions of their parent's region.
  - Subdivide as little or as much as is necessary.
  - Each internal node has exactly 4 (quadtree) or 8 (octree) children.

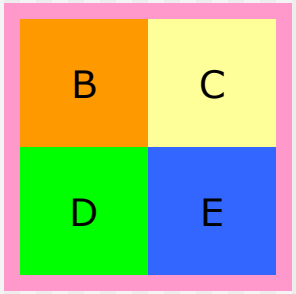
Quadtrees & Octrees: Example

- The root node of a quadtree corresponds to a square region in space.
  - Generally, this encompasses the entire "region of interest".

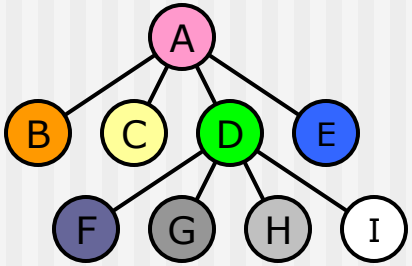


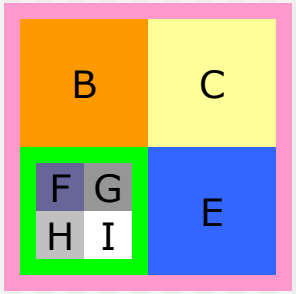
- If desired, subdivide along lines parallel to the coordinate axes, forming four smaller identically sized square regions. The child nodes correspond to these.





- Some or all of these children may be subdivided further.
- Octrees work in a similar fashion, but in 3-D, with cubical regions subdivided into 8 parts.

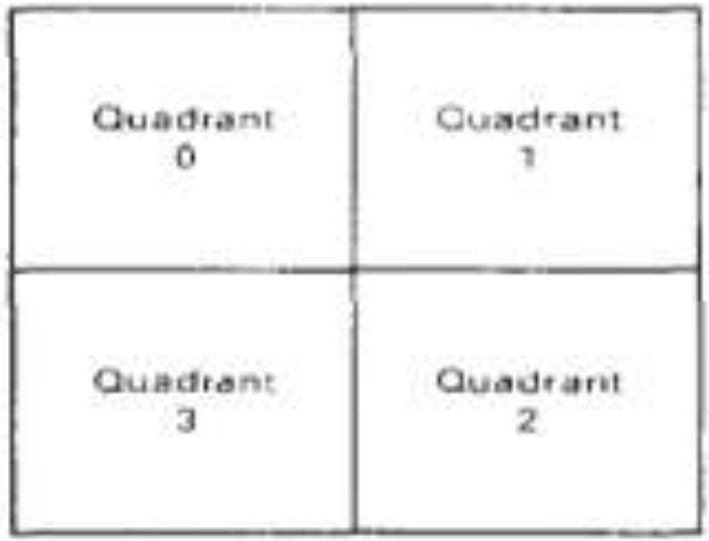




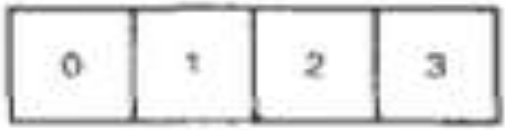
What is Quadtree

- Quadtree is a structure to represent a multi-dimension data into tree
- It contain rapid search and manipulation
- Can present a Image via use Tree structure save less storage

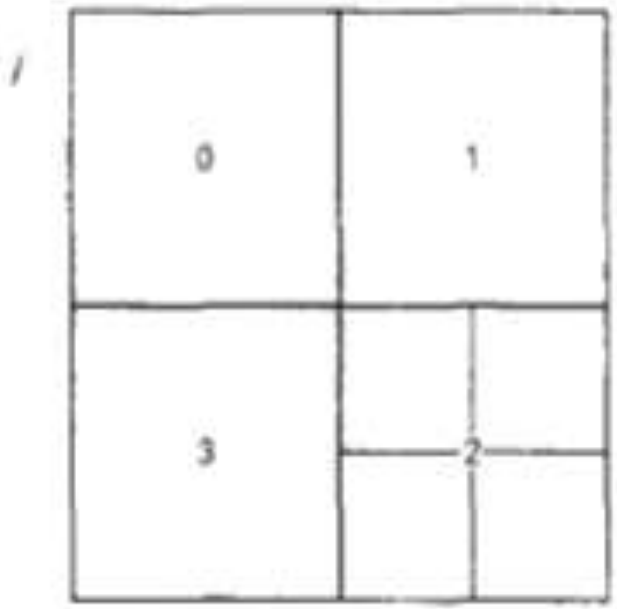
Quadtree



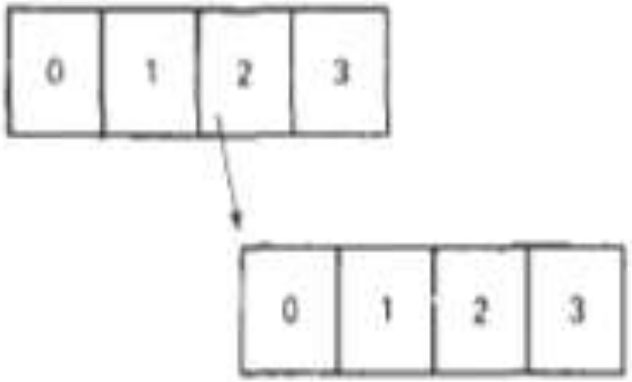
Region of a
Two Dimensional
Space



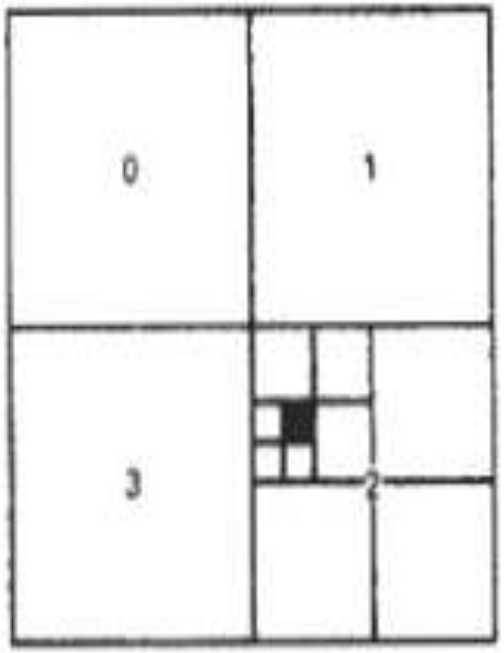
Data Elements
in the Representative
Quadtree Node

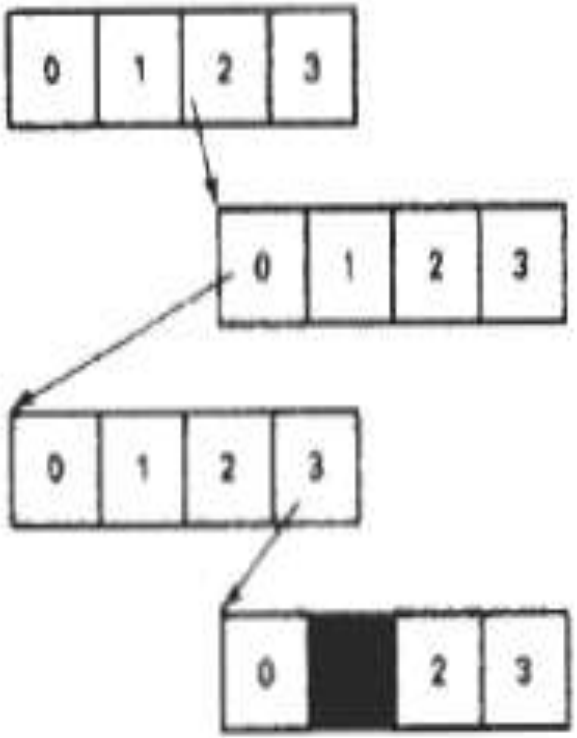


Region of a
Two-Dimensional
Space

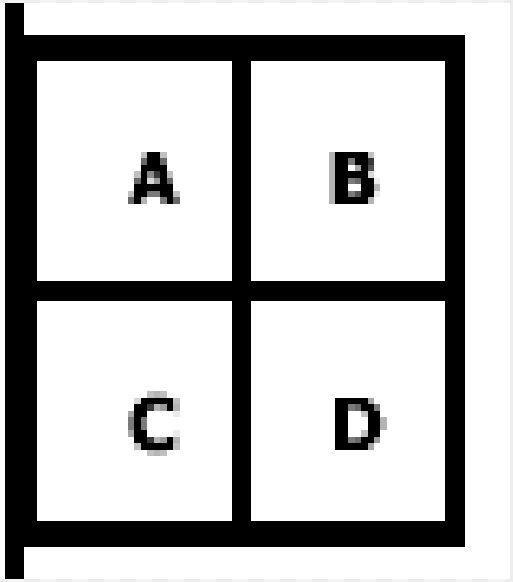


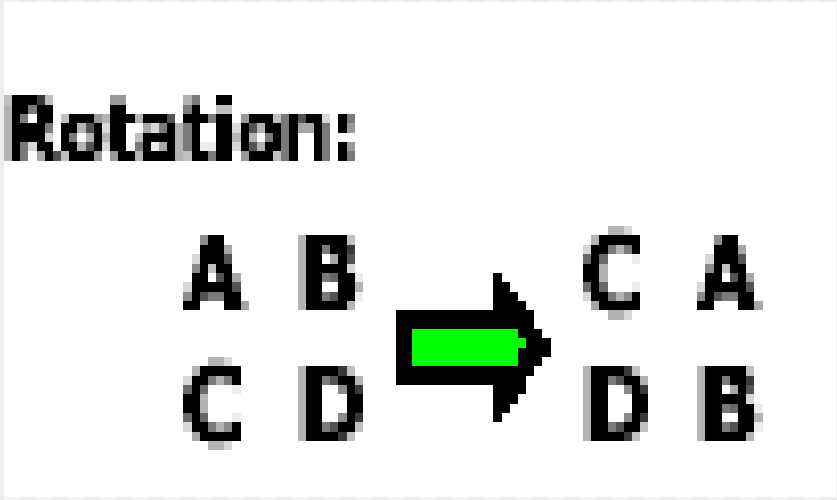
Quadtree
Representation


0
1
3
2

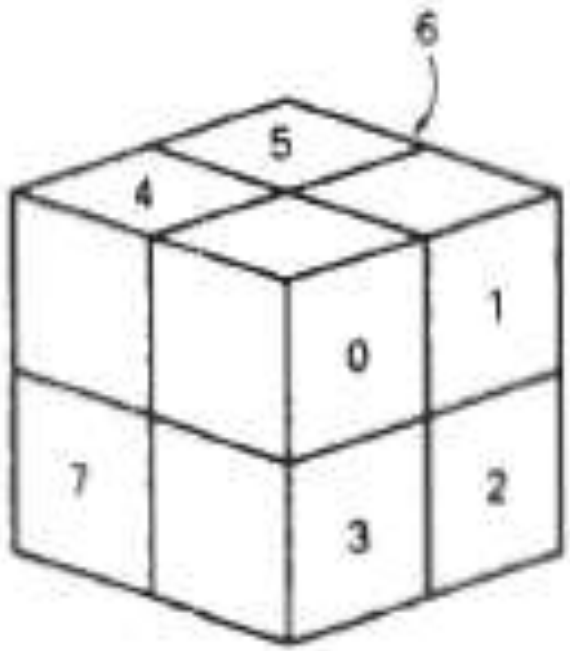

0 1 2 3
0 1 2 3
0 1 2 3
0 2 3

Manipulation (Rotate)





Octree

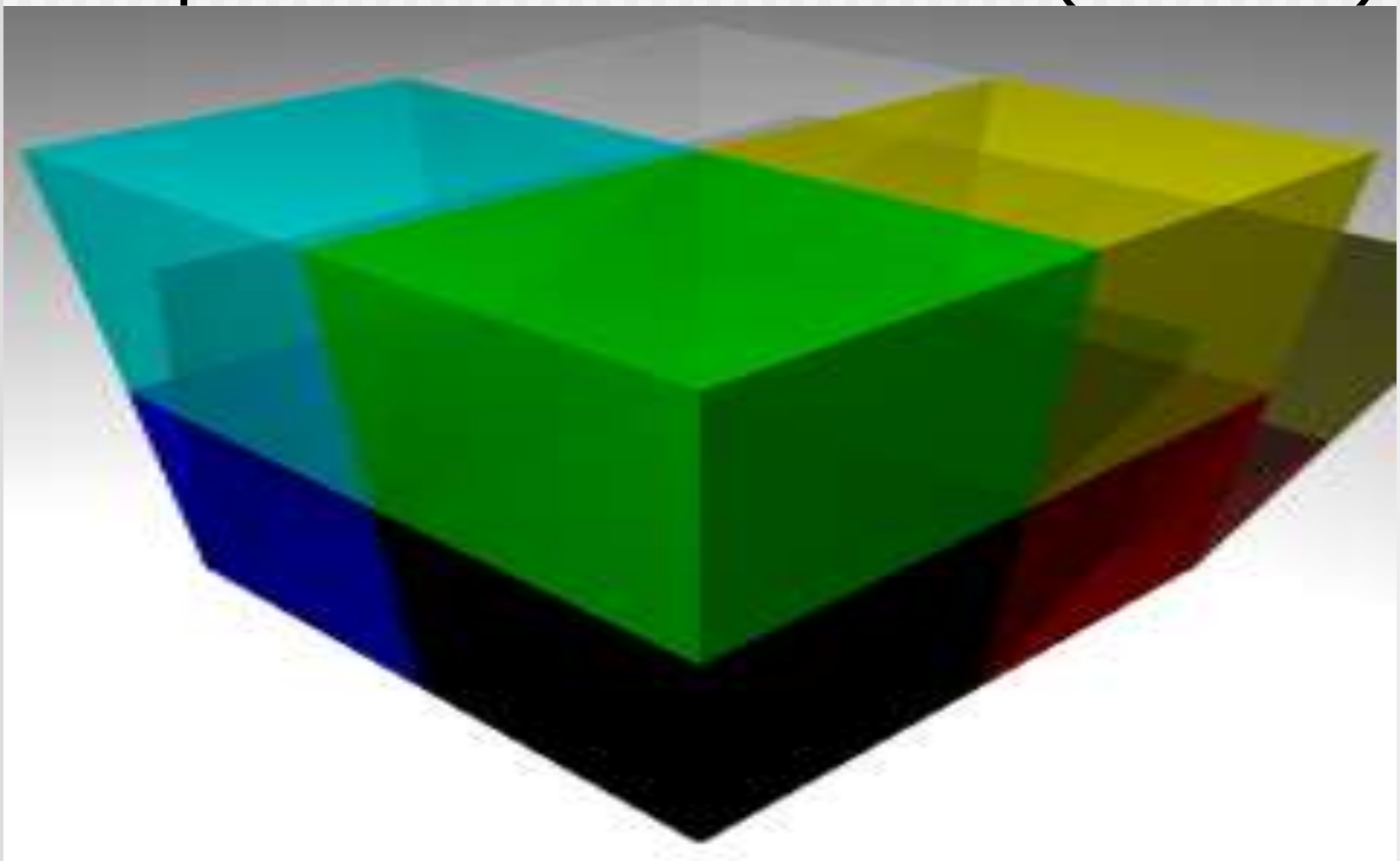


Region of a
Three-Dimensional
Space

Data Elements
in the Representative
Octree Node

Octree

- Splits into 8 subsections (octants)



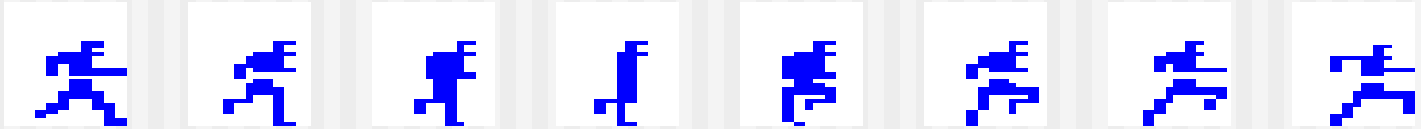
Quadtrees & Octrees: What Are They Good For?

- Handling Observer-Object Interactions
  - Subdivide the quadtree/octree until each leaf's region intersects only a small number of objects.
  - Each leaf holds a list of pointers to objects that intersect its region.
  - Find out which leaf the observer is in. We only need to test for interactions with the objects pointed to by that leaf.
- Inside/Outside Tests for Odd Shapes
  - The root node represent a square containing the shape.
  - If a node's region lies entirely inside or entirely outside the shape, do not subdivide it.
  - Otherwise, do subdivide (unless a predefined depth limit has been exceeded).
  - Then the quadtree or octree contains information allowing us to check quickly whether a given point is inside the shape.
- Sparse Arrays of Spatially-Organized Data
  - Store array data in the quadtree or octree.
  - Only subdivide if that region of space contains interesting data.
  - This is how an octree is used in the BLUIsculpt program.

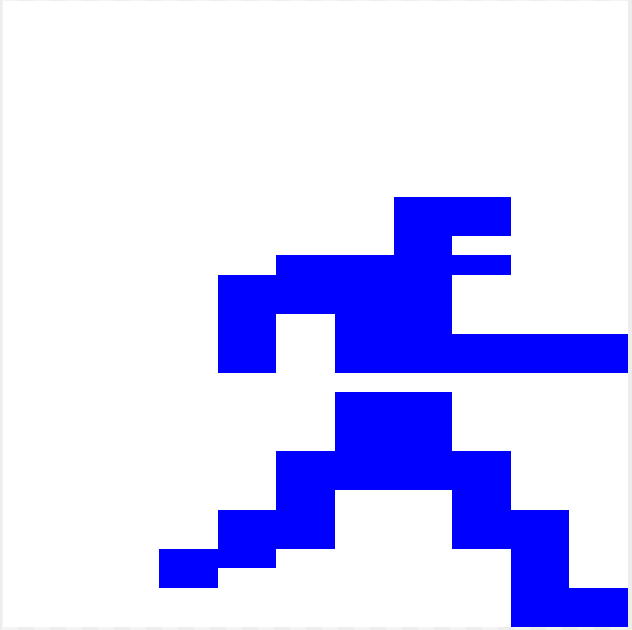
Ray Tracing

Animation

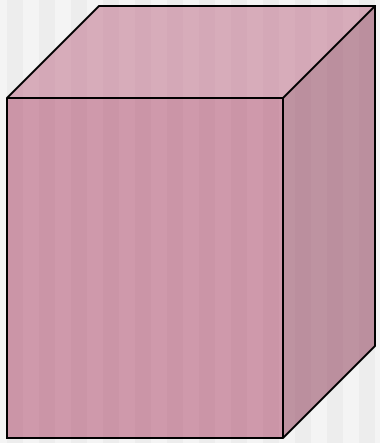
- The very first animated characters were 2D sprites.
- Just like traditional cel animation or flip books.



Animation (2)



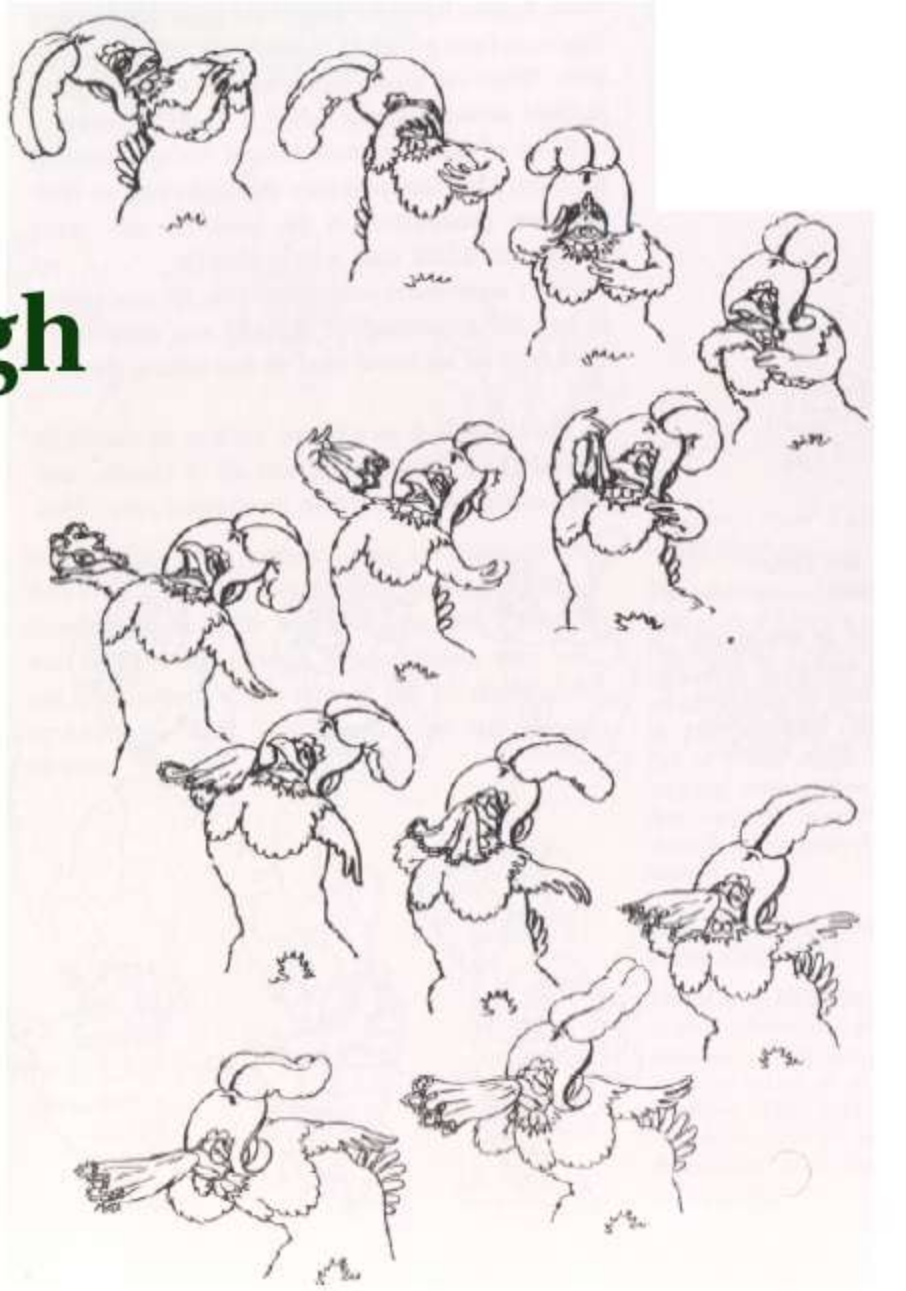
Animation: Translation



Animation: Rotation

Anticipation

Follow Through



Follow Through

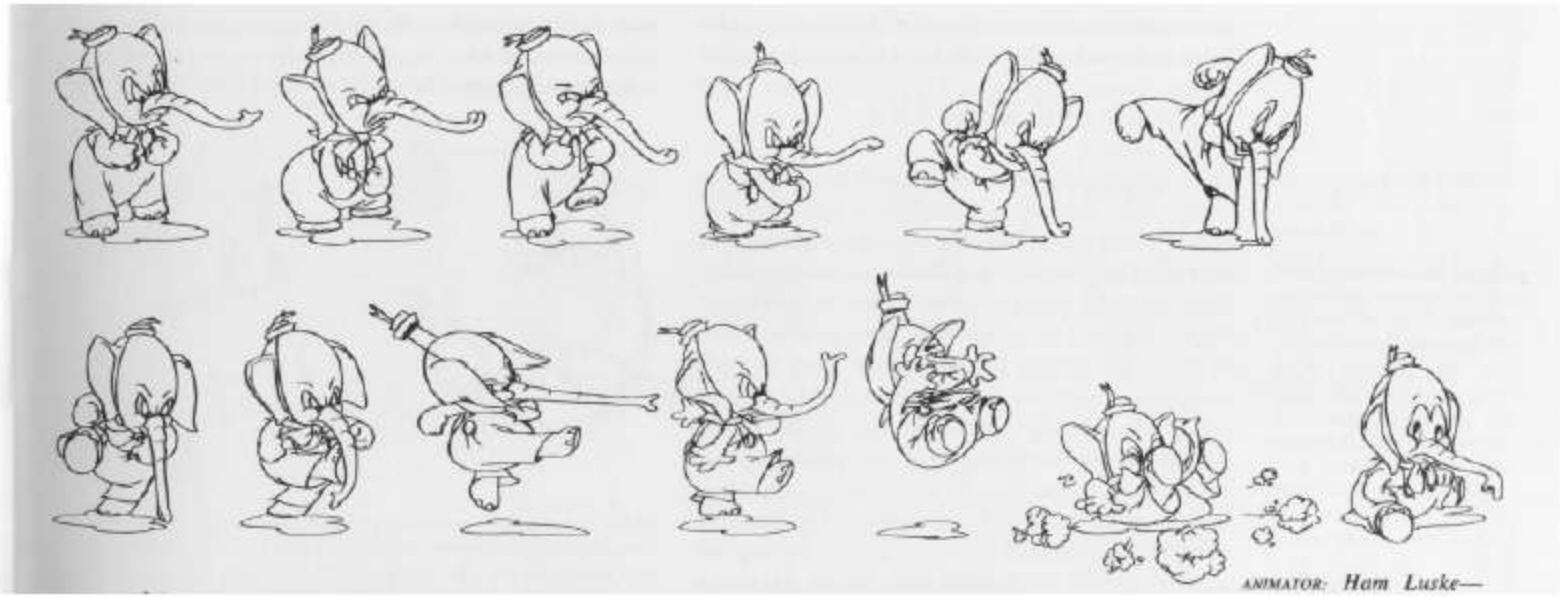


Image Transformation

- **Translation**
- **Rotation**

Morphing

- **Morphing** is a special effect in motion pictures and animations that changes (or morphs) one image into another through a seamless transition.
- Most often it is used to depict one person turning into another through technological means or as part of a fantasy or surreal sequence.

Three frames form a morph from George W. Bush to Arnold Schwarzenegger showing the mid-point between the two extremes

Animation in Video Games

presented by
Jason Gregory

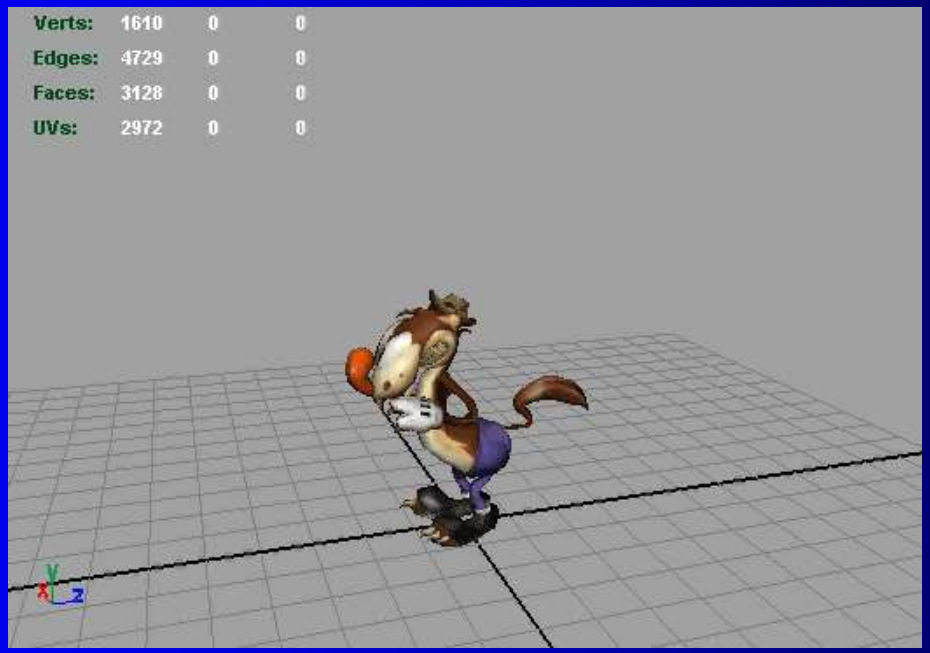
jgregory@ea.com

Agenda

- The Goal of Game Animation
- Old School Animation
- Skeletons and Skins
- How Skinning Works (Graphically)
- The Math of Skinning
- Animation: Bringing a Character to Life
- Blending and other Advanced Topics

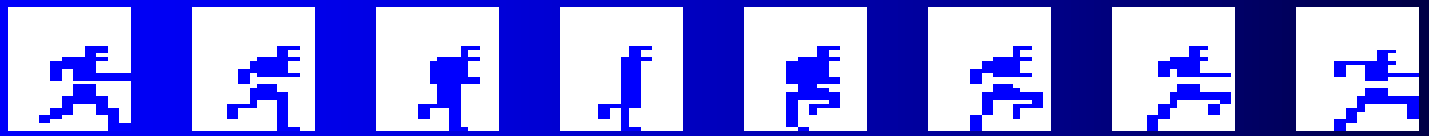
The Goal of Game Animation

- Our goal is simple: To produce realistic looking animated characters in our games!

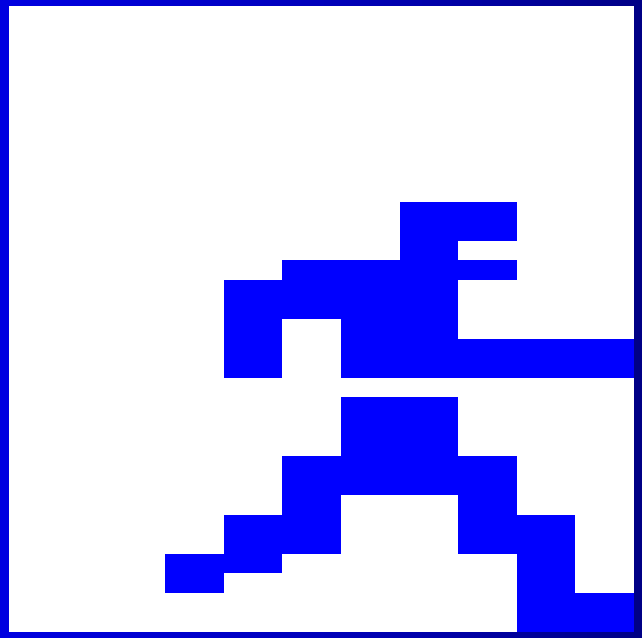


Old School Animation

- The very first animated characters were 2D sprites.
- Just like traditional cel animation or flip books.



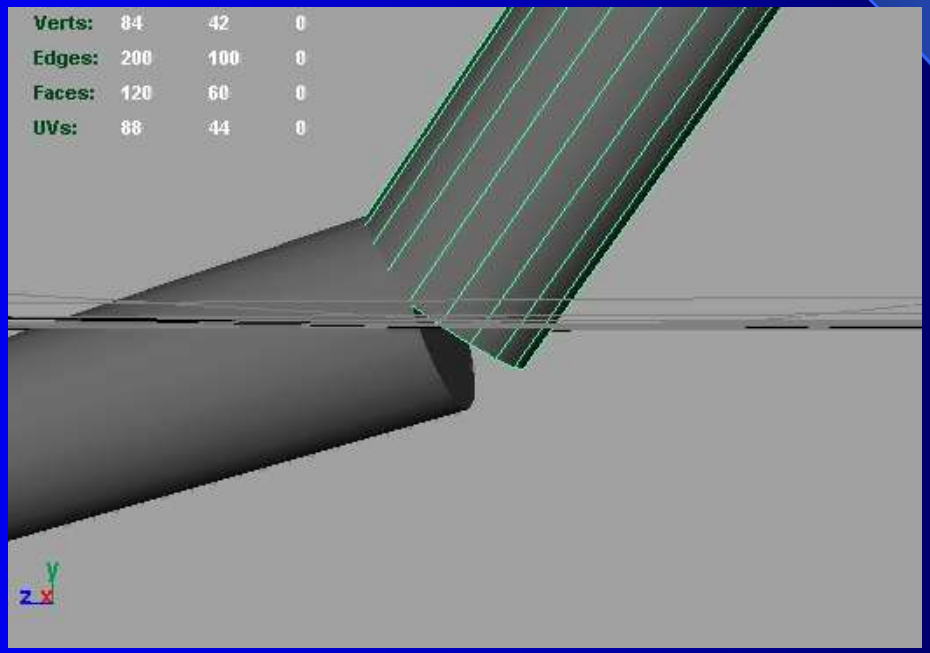
Old School Animation (2)



Old School Animation (3)

- When we moved to 3D, our first animated characters were “jointed.”
- Each limb or part of a limb is a separate rigid object.
- Problem: Interpenetration at joints!

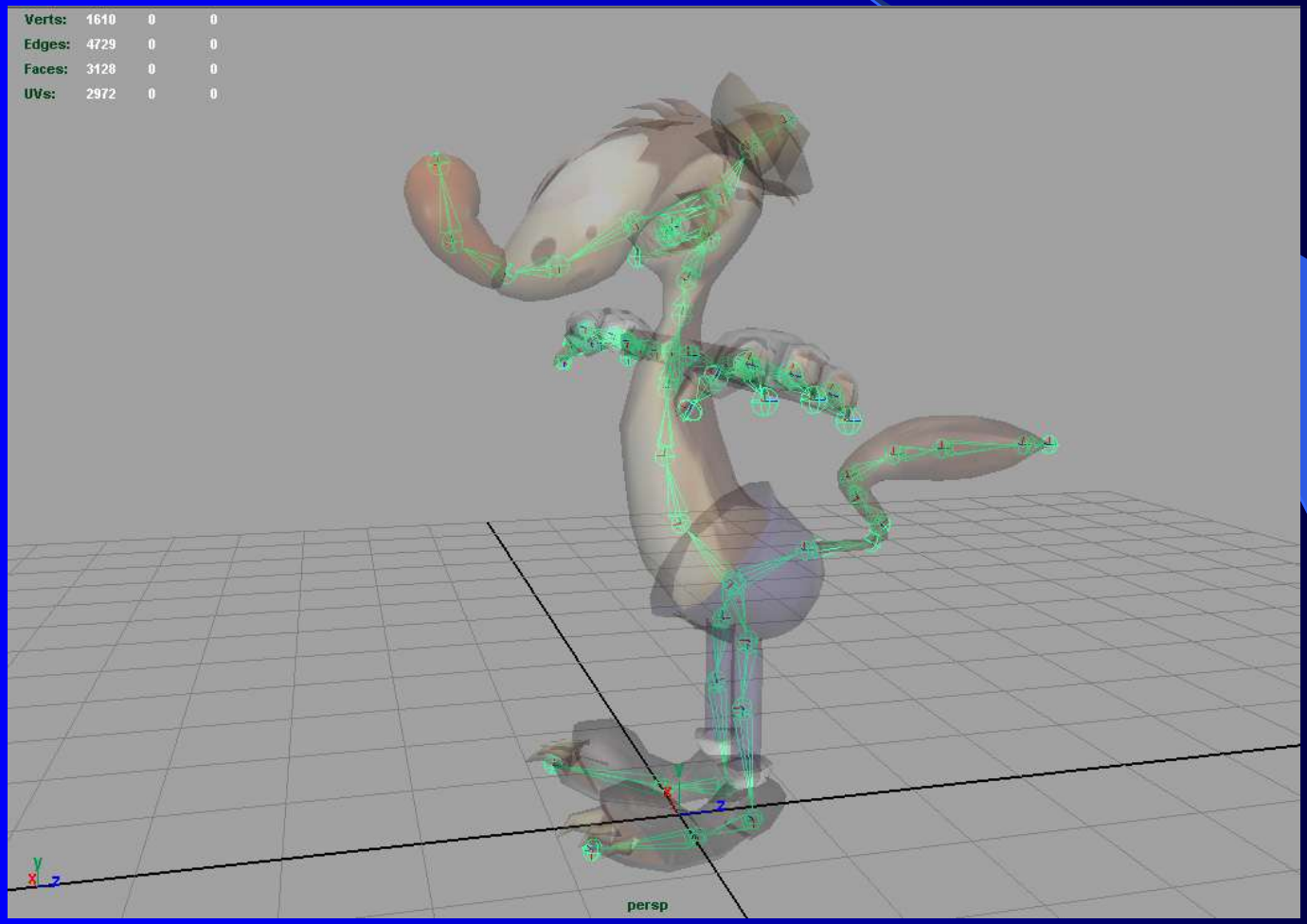
Old School Animation (4)



Skeletons and Skins

- Modern approach is called “skinning.”
- Basic idea:
  - Create a jointed skeleton.
  - Attach the skin to the skeleton.
  - Move skeleton around – skin follows it.
- Skin is a 3D model made out of triangles.
- Skeleton is invisible – only the skin is seen by the player.

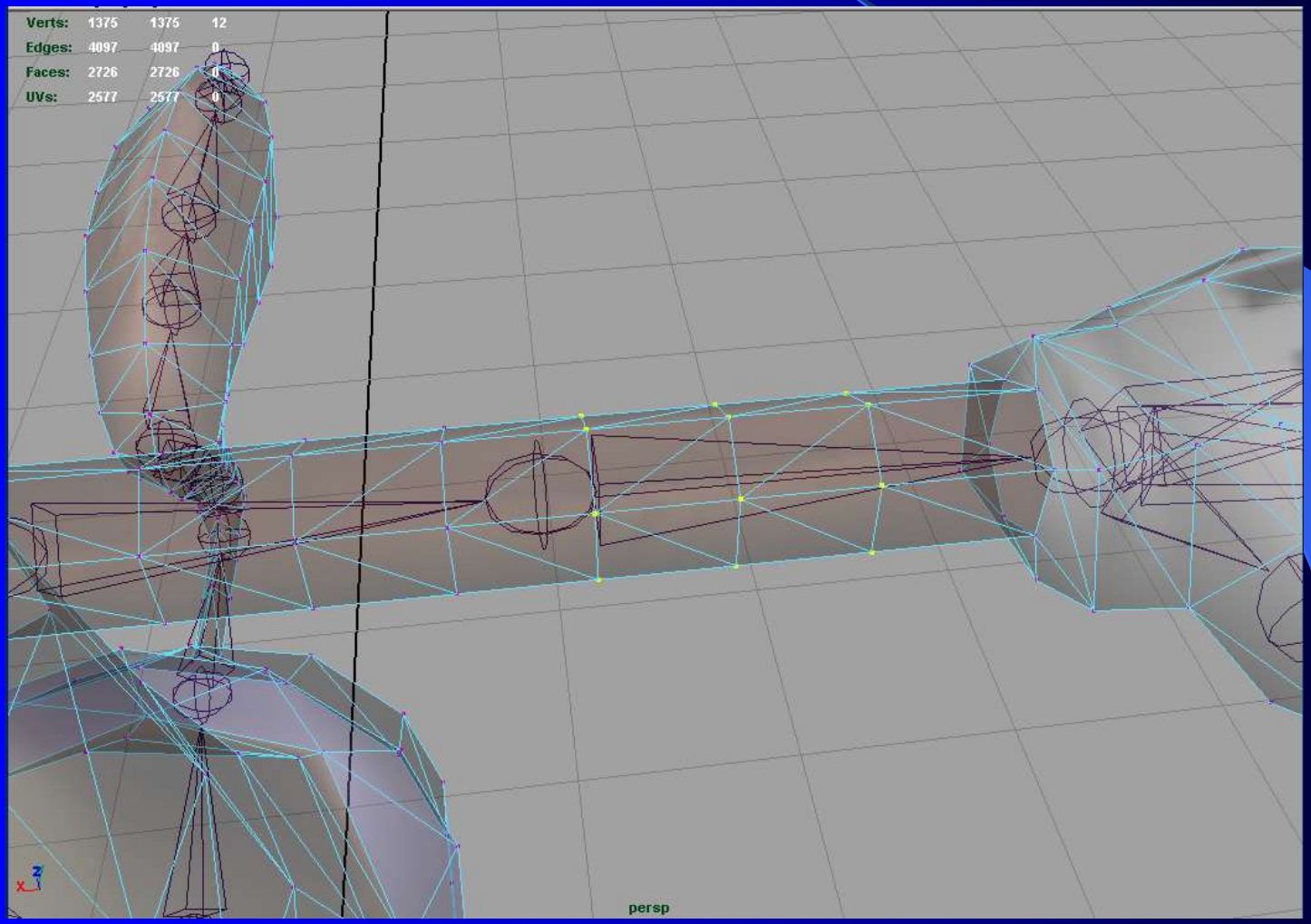
Skeletons and Skins (2)



Skeletons and Skins (3)

- Each vertex of each triangle is attached to one or more bones.
  - We use **weights** to define bones' influences.
  - Weights at a joint must always add up to 1.

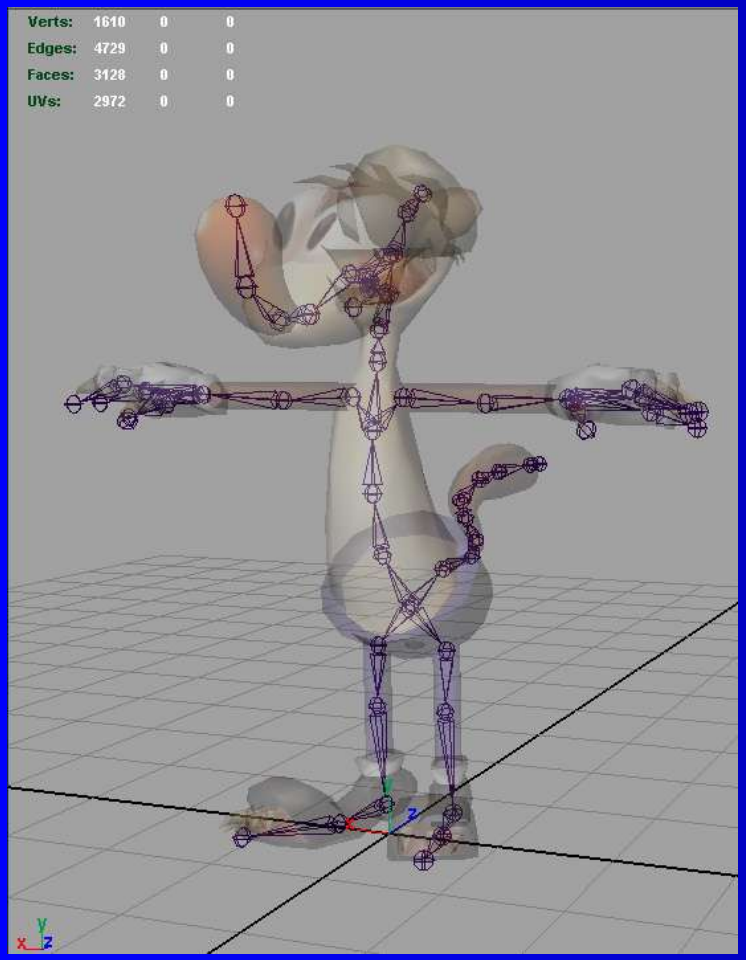
Skeletons and Skins (4)



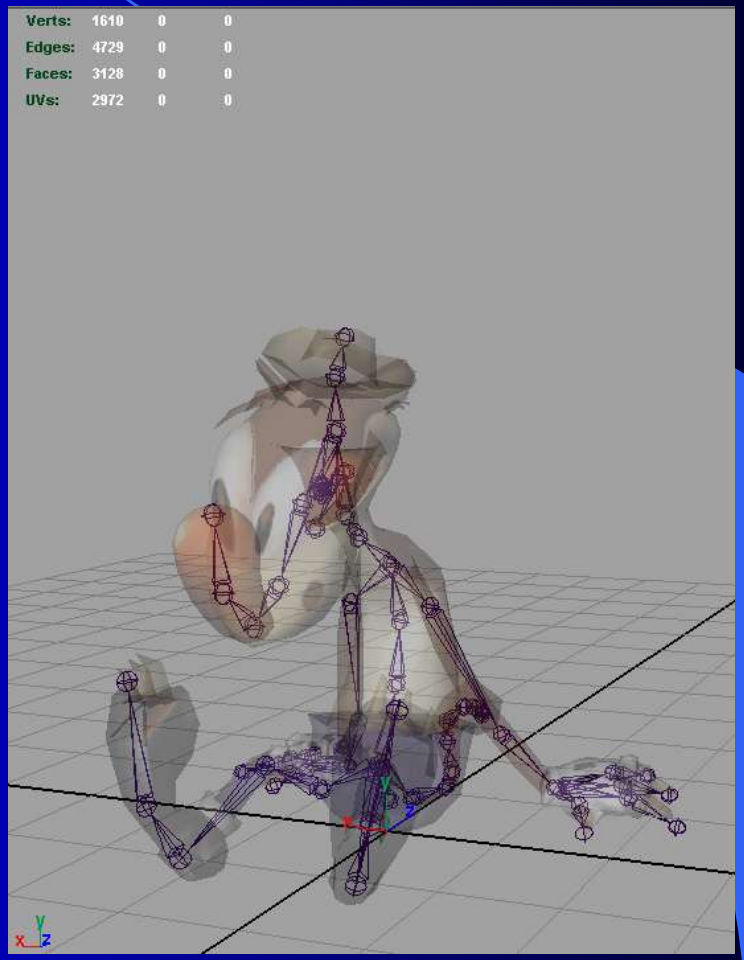
Skeletons and Skins (5)

- Skeletons have two kinds of poses:
  - **Bind Pose:** The skeleton's pose when the skin was first attached.
  - **Current Pose:** Any other pose of the skeleton; usually a frame of an animation.
- The **bind pose** is like a "home base" for the character's skeleton.
- If you drew the mesh without its skeleton, it would appear in its bind pose.

Skeletons and Skins (6)



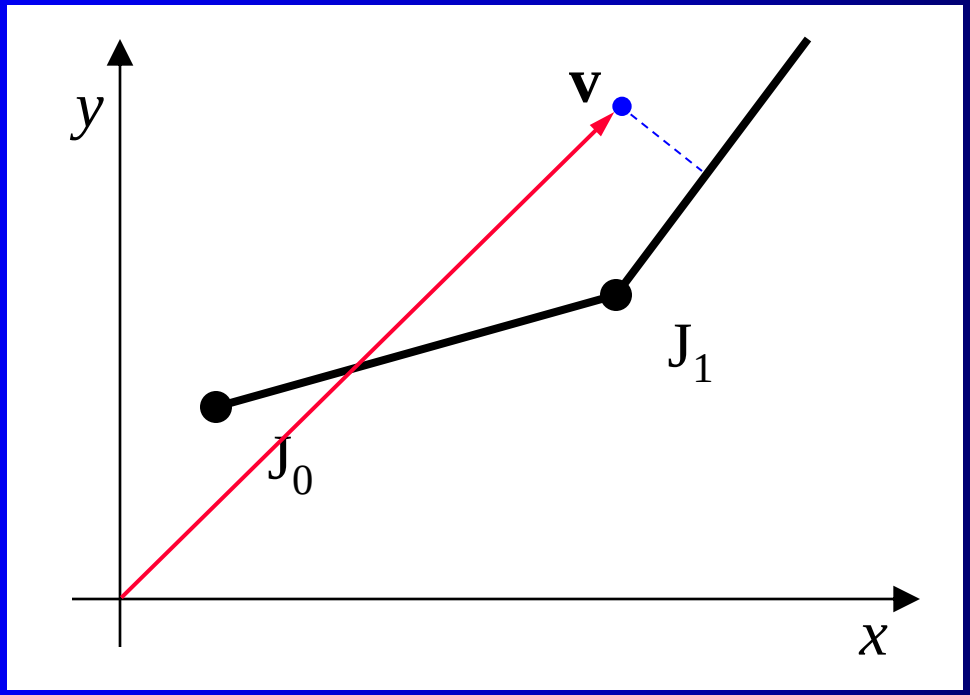
Bind Pose



Current Pose

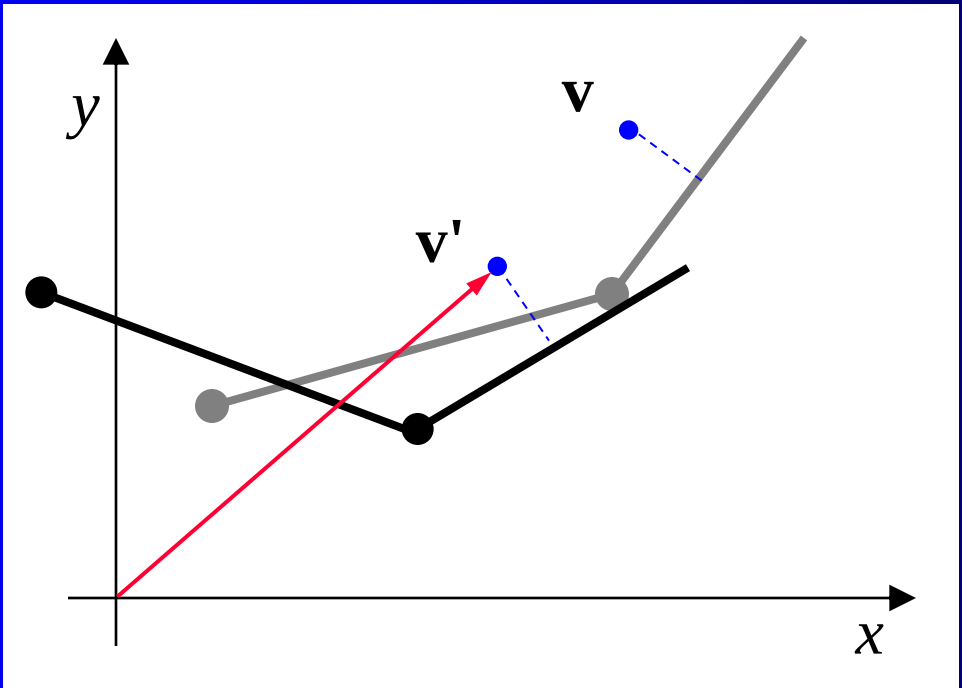
How Skinning Works

- Consider a single vertex (**v**) skinned to the joint J_1. The skeleton is in **bind pose**:



How Skinning Works (2)

- We want to find the vertex's new location (**v'**) in the **current pose**.

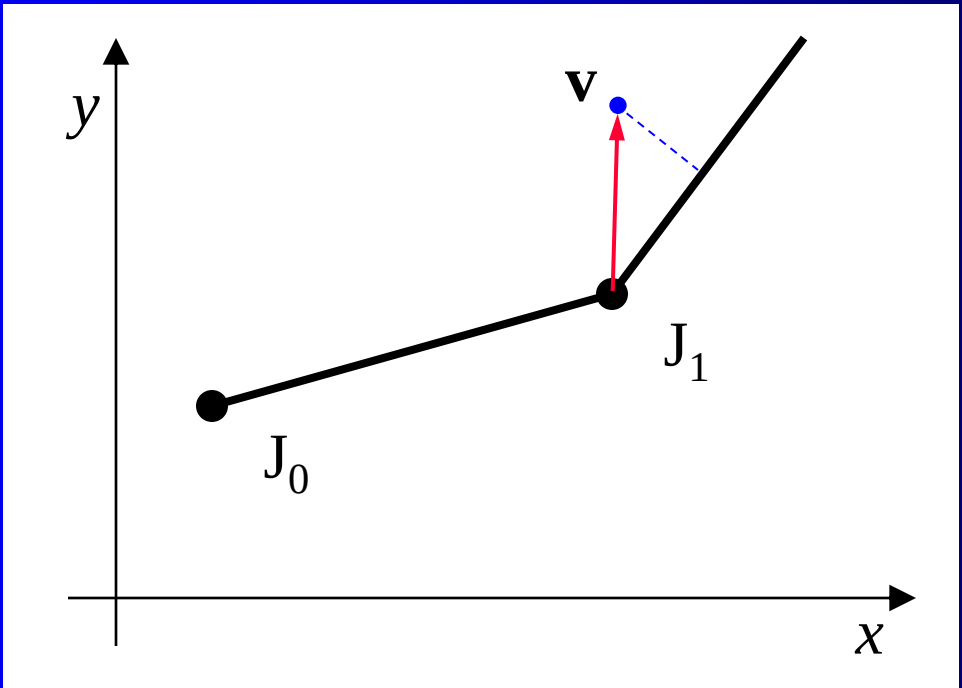


How Skinning Works (3)

- The basic idea is to transform the vertex:
  - from **model space**
  - into **joint space**
- The coordinates of the vertex are *invariant* in joint space!
  - So, we can move the joint around all we want.
- When we're done, we go back to **model space** to find the final position (**v**').

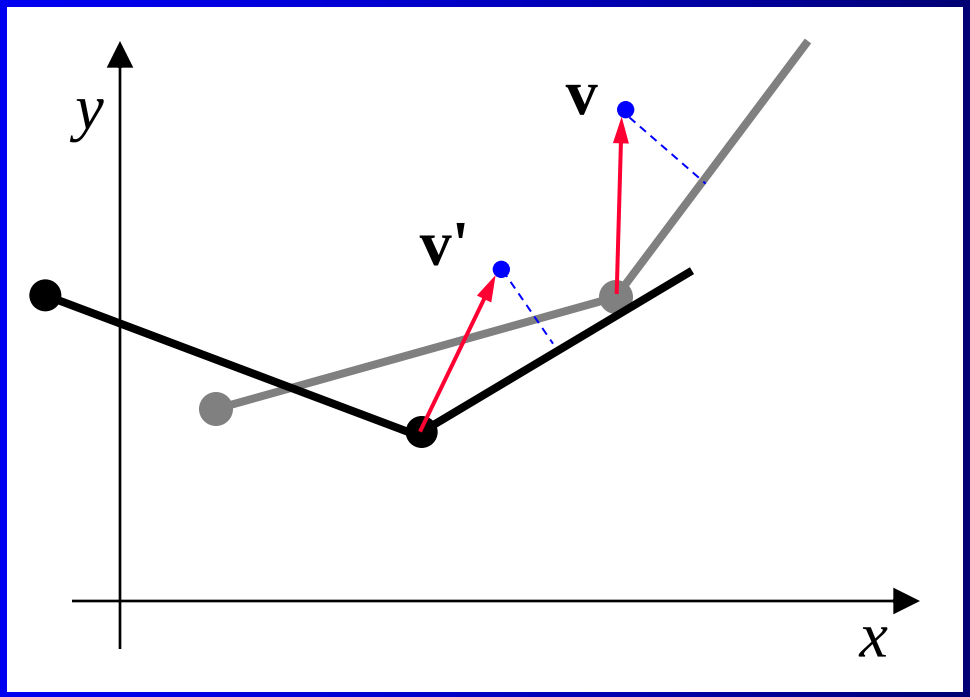
How Skinning Works (4)

- Here's the original vertex (**v**), but now in the joint space of J_1:



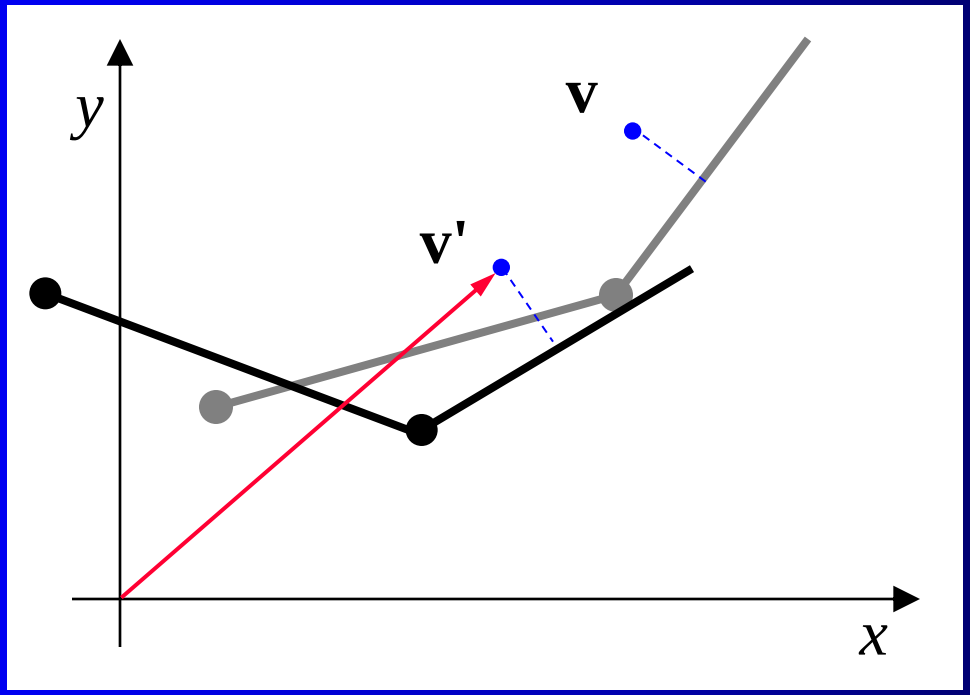
How Skinning Works (5)

- No matter what pose the skeleton is in, **v** and **v'** are the same when in **joint space**.



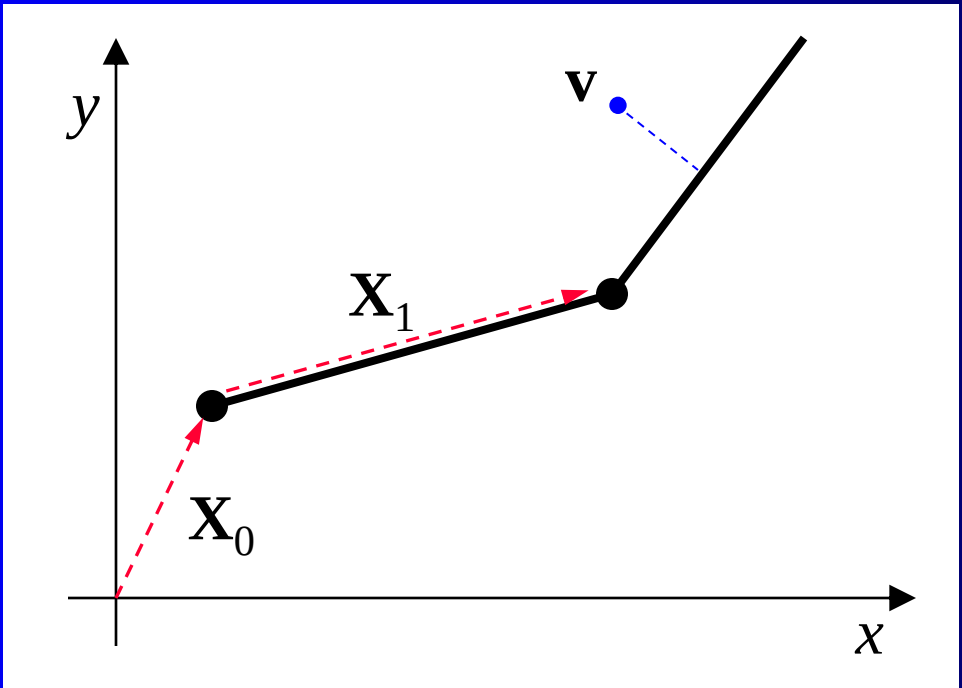
How Skinning Works (6)

- Finally, we go back to model space to find the final location of the vertex (**v**').



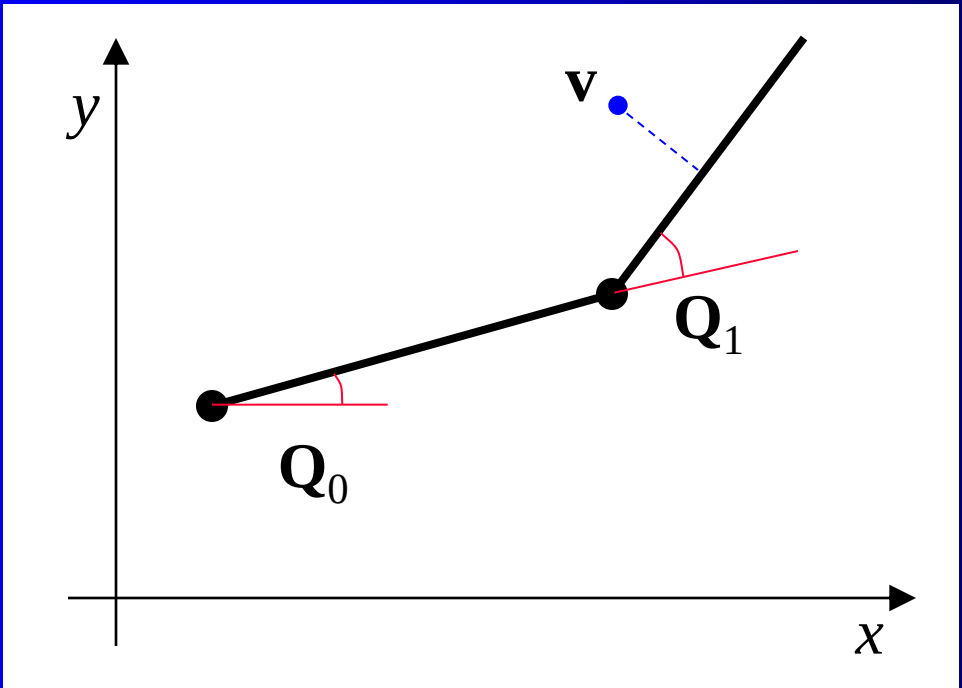
The Math of Skinning

- Let $\mathbf{X}_i$ be the translation of joint i.



The Math of Skinning (2)

- Let $\mathbf{Q}_i$ be the rotation of joint i.



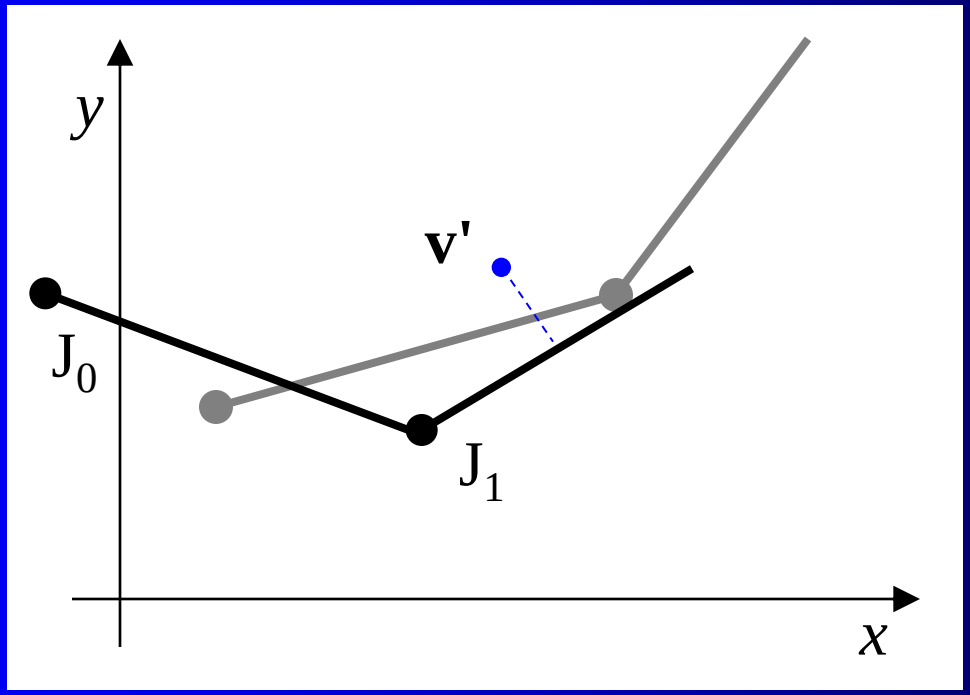
The Math of Skinning (3)

- We describe the **bind pose matrix** of joint J_1 as the matrix product of all the translations and rotations from the root joint to the joint in question:

$$\mathbf{B}_j = \prod_{i=0}^{j} \mathbf{X}_i \mathbf{Q}_i$$

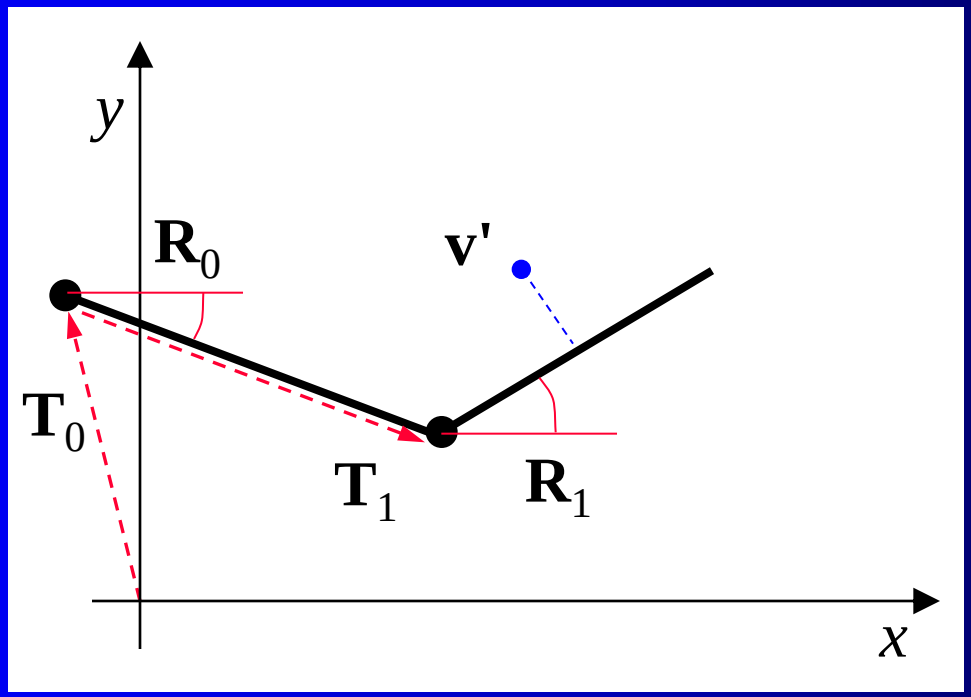
The Math of Skinning (4)

- Now consider what happens when we move the skeleton into the current pose:



The Math of Skinning (5)

- This time, let $\mathbf{T}_i$ be the translation of joint i, and let $\mathbf{R}_i$ be the rotation of joint i:



The Math of Skinning (6)

- The matrix describing the **current pose** is:

$$\mathbf{P}_j = \prod_{i=0}^{j} \mathbf{T}_i \mathbf{R}_i$$

which is similar to the **bind pose** matrix:

$$\mathbf{B}_j = \prod_{i=0}^{j} \mathbf{X}_i \mathbf{Q}_i$$

The Math of Skinning (7)

- We multiply **v** by $\mathbf{B}^{-1}$ to get it into joint space from the **bind pose**.
- Then we multiply *that* by **P** to get it back into model space, in the **current pose**.

The Math of Skinning (8)

- Mathematically, this is:

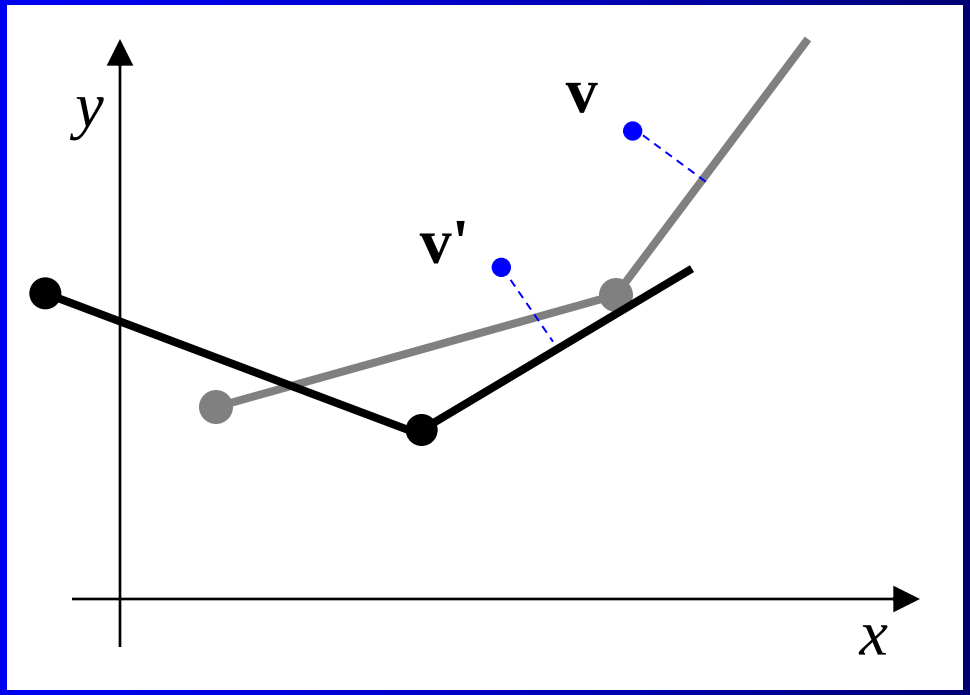
$$\mathbf{v}' = \left(\prod_{i=0}^{j} \mathbf{P}_i \right) \left(\prod_{i=0}^{j} \mathbf{B}_i \right)^{-1} \mathbf{v}$$

$$\mathbf{v}' = \left(\prod_{i=0}^{j} \mathbf{P}_i \right) \left(\prod_{i=j}^{0} \mathbf{B}_i^{-1} \right) \mathbf{v}$$

$$\mathbf{v}' = \mathbf{P}_0 \mathbf{P}_1 \mathbf{B}_1^{-1} \mathbf{B}_0^{-1} \mathbf{v}$$

The Math of Skinning (9)

- Voila! We can find **v'** for any pose imaginable!



The Math of Skinning (10)

- We do these calculations on each and every vertex in the model.
- Then we draw the final vertices.
- For vertices that are affected by *more than one* joint, we take a **weighted average** of the positions due to each joint.

The Math of Skinning (11)

- The weighted average for a vertex affected by joints *j* and *k* would be:

$$\mathbf{v}'_j = \left(\prod_{i=0}^{j} \mathbf{P}_i\right)\left(\prod_{i=0}^{j} \mathbf{B}_i\right)^{-1} \mathbf{v}$$

$$\mathbf{v}'_k = \left(\prod_{i=0}^{k} \mathbf{P}_i\right)\left(\prod_{i=0}^{k} \mathbf{B}_i\right)^{-1} \mathbf{v}$$

$$\mathbf{v}' = w_j \mathbf{v}'_j + w_k \mathbf{v}'_k$$

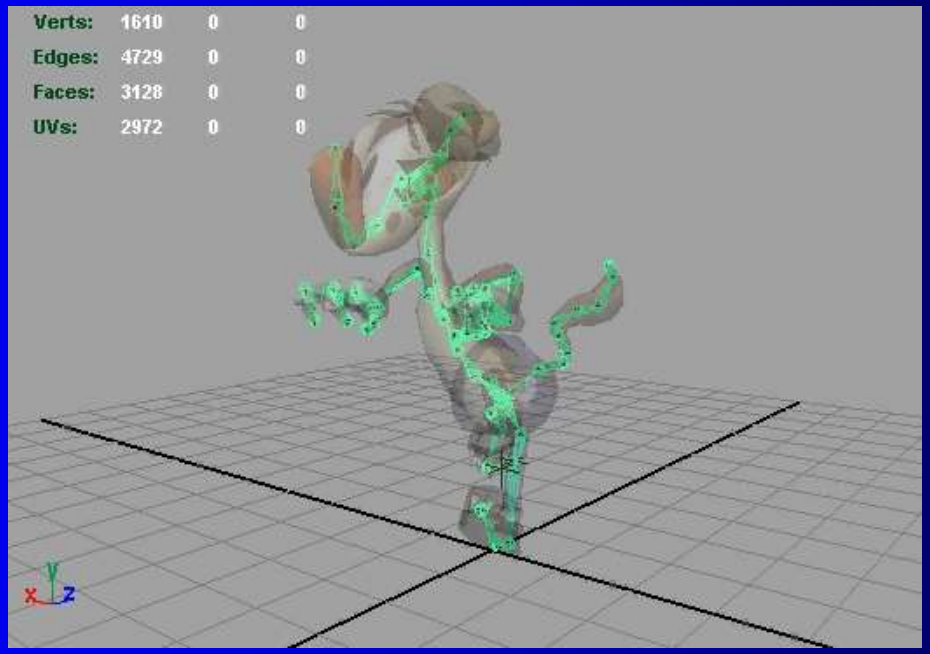
$$\text{where } w_j + w_k = 1$$

Animation: Bringing Characters to Life

- An **animation** is really just a sequence of **poses** at various points in time. The poses are called **keys**.
- An animation can be described mathematically as:
  $\{ \mathbf{P}_j(t) \} \; \forall \; j$
  i.e. a set of pose matrices (keys) for all joints *j*, each of which is a function of time *t*.
- To play back the animation, we **extract a pose** at the current time index, **skin** the model to that pose, and then **draw** the model.

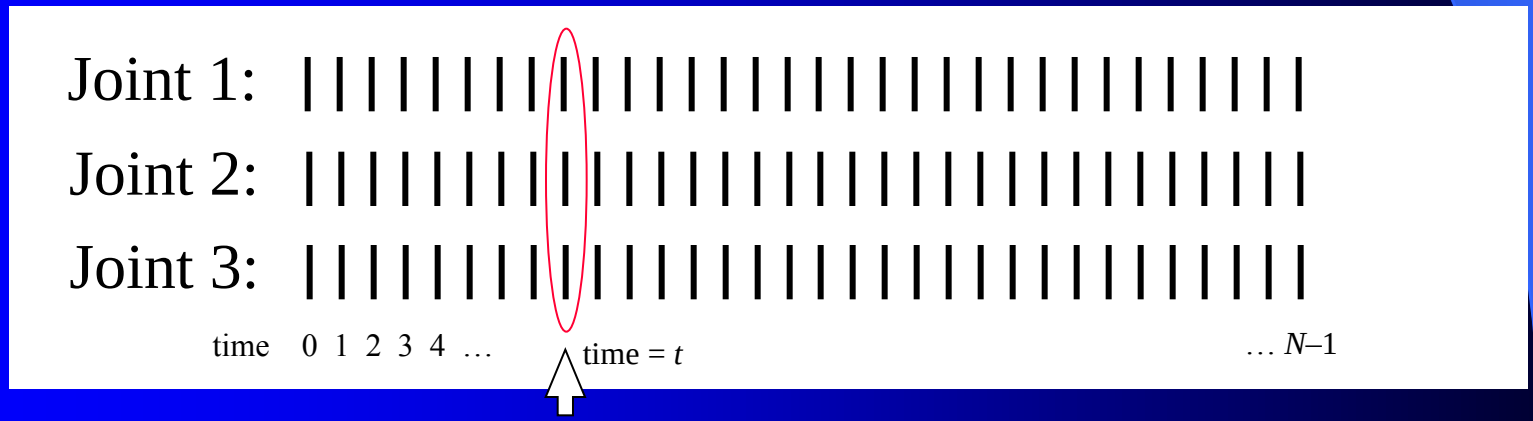
Bringing Characters to Life (2)

- Run Weasel, run!



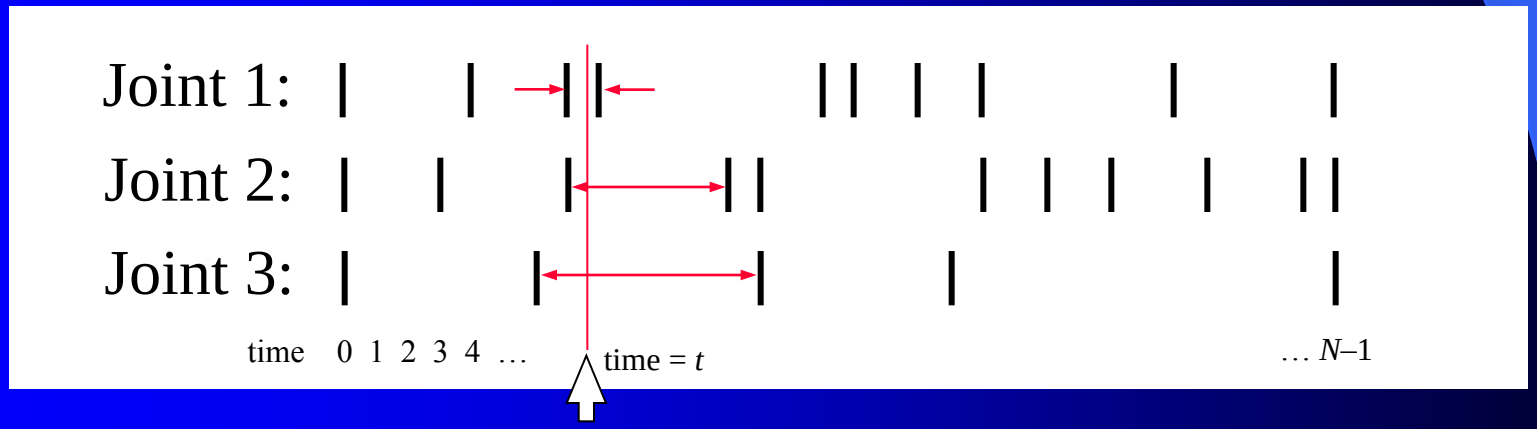
Bringing Characters to Life (3)

- In a simple animation system, the keys are *equally spaced* in time.
- If we further restrict ourselves to integer time indices, then extracting a pose amounts to selecting the appropriate key.



Bringing Characters to Life (3)

- To reduce memory overhead, the keys can be **compressed**, and might **not** be uniformly spaced.
- We will want to allow the time index to be a **real number** (*i.e.* floating-point).
- So, extracting a pose now requires **interpolation** between adjacent key frames.



Interpolation and Blending

- To interpolate **positions**, we use simple vector linear interpolation (LERP).

$$\mathbf{r} = (1-\beta)\mathbf{r}_1 + \beta\mathbf{r}_2$$

i.e.

$$r_x = (1-\beta)r_{1x} + \beta r_{2x}$$
$$r_y = (1-\beta)r_{1y} + \beta r_{2y}$$
$$r_z = (1-\beta)r_{1z} + \beta r_{2z}$$

Interpolation and Blending (2)

- To interpolate **rotations**, we must use **quaternions**. (It is next to impossible to interpolate matrices.)
- We have two choices when interpolating quats:
  - Linear interpolation (LERP)
  - Spherical linear interpolation (SLERP)

Interpolation and Blending (3)

- A quaternion LERP is identical to a vector LERP, but with 4 components.
- SLERP is like a LERP, but the weights are no longer (1-β) and β. Instead they are:

$$c = \cos\theta = \mathbf{r}_1 \cdot \mathbf{r}_2 \qquad \therefore \theta = \cos^{-1} c$$

$$s = \sin\theta$$

$$w_1 = \frac{\sin((1-\beta)\theta)}{s} \qquad w_2 = \frac{\sin(\beta\theta)}{s}$$

$$\mathbf{r} = w_1\mathbf{r}_1 + w_2\mathbf{r}_2$$

Interpolation and Blending (4)

- LERP and SLERP can be used to interpolate between adjacent key frames for a specific time *t*.
- Interpolation can also be used to blend two **entirely different** animations together!
- For example, instead of a character being able to walk ***or*** run, he can do anything in between!
- The blend factor controls how much **walk** and how much **run** we see.

  β=**0**: full walk β=**1**: full run
  β=**0.5**: half walk, half run

The Animation Pipeline

- Typical animation pipeline:
  - Pose extraction at current time *t*
  - Pose blending
  - Matrix palette generation
  - Palette-driven rendering
- The matrix palette maps directly to modern vertex shader architectures (a.k.a. indexed skinning).

Advanced Topics

- Key frame **compression** techniques
- Representing animations as **spline curves** instead of interpolated key frames
- Action state machines
- Skeletal partitioning
- Rag-doll physics
- …

Q&A

- Thanks for your attention!
- Questions can also be sent to:

Jason Gregory
Electronic Arts Los Angeles
jgregory@ea.com

Animation

- Making things alive/Making them move
- Traditional Animation
  - Interpolating between key frames
- Kinematics
- Dynamics
- Motion Capture
- Behaviors

Traditional Cel Animation

- Film runs at 24 frames per second (fps)
  - That's 1440 pictures to draw per minute
  - 1800 fpm for video (30fps)
- Productions issues:
  - Need to stay organized for efficiency and cost reasons
  - Need to render the frames systematically (render farms)
- Artistic issues:
  - How to create the desired look and mood while conveying story?
  - Artistic vision has to be converted into a sequence of still frames
  - Not enough to get the stills right--must look right at full speed
    - » Hard to "see" the motion given the stills
    - » Hard to "see" the motion at the wrong frame rate

(Pollard http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/slides/25-animII.pdf)

Traditional Animation: The Process

- Story board
  - Sequence of drawings with descriptions
  - Story-based description
- Key Frames
  - Draw a few important frames as line drawings
    - » For example, beginning of stride, end of stride
- Inbetweens
  - Draw the rest of the frames
- Painting
  - Redraw onto acetate *Ce/s*, color them in

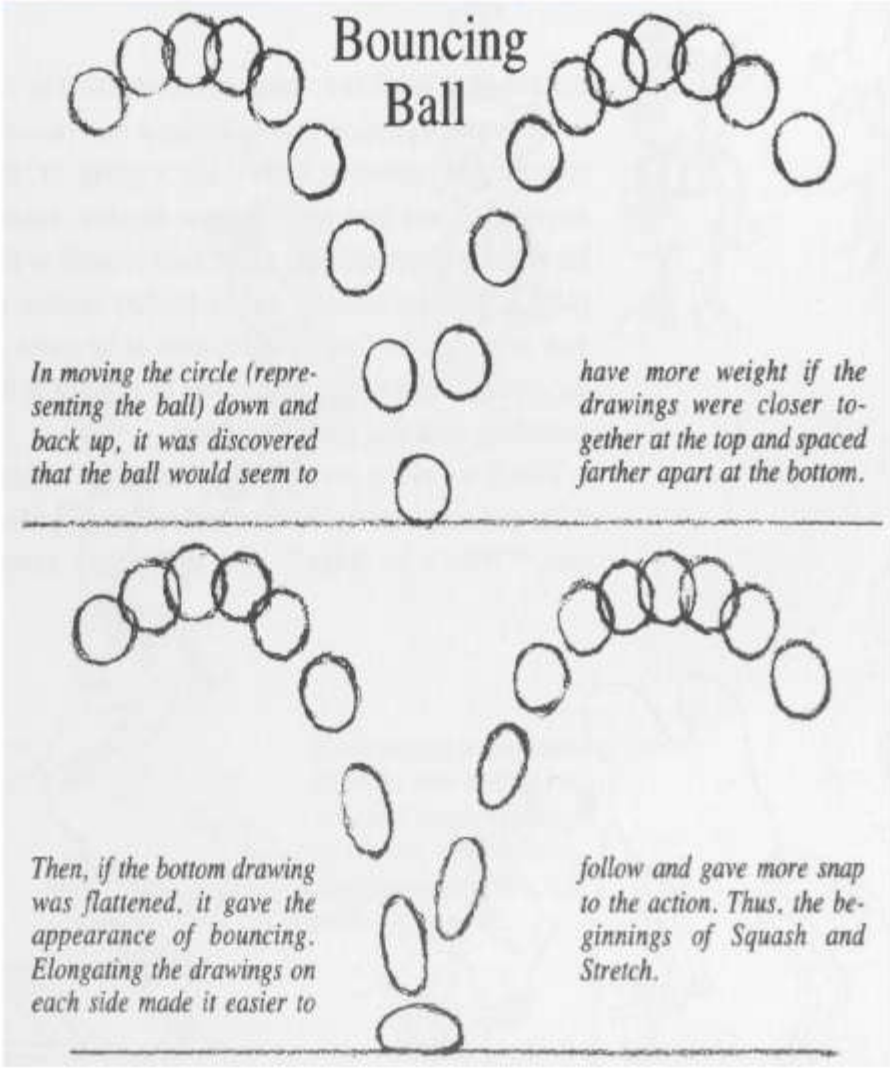
(Pollard http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/slides/25-animII.pdf)

Layered Motion

- It's often useful to have multiple layers of animation
  - How to make an object move in front of a background?
  - Use one layer for background, one for object
  - Can have multiple animators working simultaneously on different layers, avoid re-drawing and flickering
- Transparent acetate allows multiple *layers*
  - Draw each separately
  - Stack them together on a copy stand
  - Transfer onto film by taking a photograph of the stack

(Pollard http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/slides/25-animII.pdf)

Squash and Stretch



(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/principles.pdf)

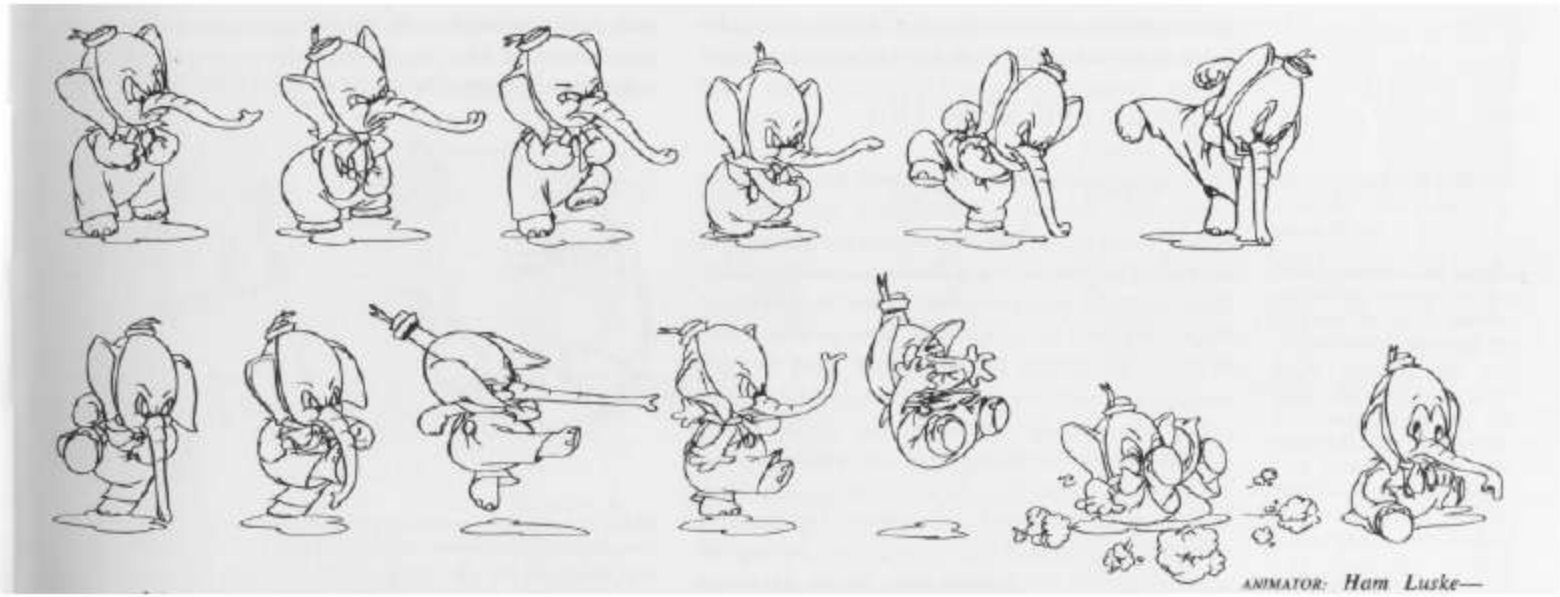
Anticipation

(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/principles.pdf)

Follow Through

(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/principles.pdf)

Follow Through



(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/principles.pdf)

Cartoon Laws of Physics

Authorship Unknown

Cartoon Law I

Any body suspended in space will remain in space until made aware of its situation. Daffy Duck steps off a cliff, expecting further pastureland. He loiters in midair, soliloquizing flippantly, until he chances to look down. At this point, the familiar principle of 32 feet per second per second takes over.

Cartoon Law II

Any body in motion will tend to remain in motion until solid matter intervenes suddenly. Whether shot from a cannon or in hot pursuit on foot, cartoon characters are so absolute in their momentum that only a telephone pole or an outsize boulder retards their forward motion absolutely. Sir Isaac Newton called this sudden termination of motion the stooge's surcease.

Cartoon Law III

Any body passing through solid matter will leave a perforation conforming to its perimeter. Also called the silhouette of passage, this phenomenon is the specialty of victims of directed-pressure explosions and of reckless cowards who are so eager to escape that they exit directly through the wall of a house, leaving a cookie-cutout-perfect hole. The threat of skunks or matrimony often catalyzes this reaction.

Cartoon Law IV

The time required for an object to fall twenty stories is greater than or equal to the time it takes for whoever knocked it off the ledge to spiral down twenty flights to attempt to capture it unbroken. Such an object is inevitably priceless, the attempt to capture it inevitably unsuccessful.

Cartoon Law V

All principles of gravity are negated by fear. Psychic forces are sufficient in most bodies for a shock to propel them directly away from the earth's surface. A spooky noise or an adversary's signature sound will induce motion upward, usually to the cradle of a chandelier, a treetop, or the crest of a flagpole. The feet of a character who is running or the wheels of a speeding auto need never touch the ground, especially when in flight.

Cartoon Law VI

As speed increases, objects can be in several places at once. This is particularly true of tooth-and-claw fights, in which a character's head may be glimpsed emerging from the cloud of altercation at several places simultaneously. This effect is common as well among bodies that are spinning or being throttled. A `wacky' character has the option of self-replication only at manic high speeds and may ricochet off walls to achieve the velocity required.

Cartoon Law VII

Certain bodies can pass through solid walls painted to resemble tunnel entrances; others cannot. This trompe l'oeil inconsistency has baffled generations, but at least it is known that whoever paints an entrance on a wall's surface to trick an opponent will be unable to pursue him into this theoretical space. The painter is flattened against the wall when he attempts to follow into the painting. This is ultimately a problem of art, not of science.

Cartoon Law VIII

Any violent rearrangement of feline matter is impermanent. Cartoon cats possess even more deaths than the traditional nine lives might comfortably afford. They can be decimated, spliced, splayed, accordion-pleated, spindled, or disassembled, but they cannot be destroyed. After a few moments of blinking self pity, they reinflate, elongate, snap back, or solidify. Corollary: A cat will assume the shape of its container.

Cartoon Law IX

Everything falls faster than an anvil.

Cartoon Law X

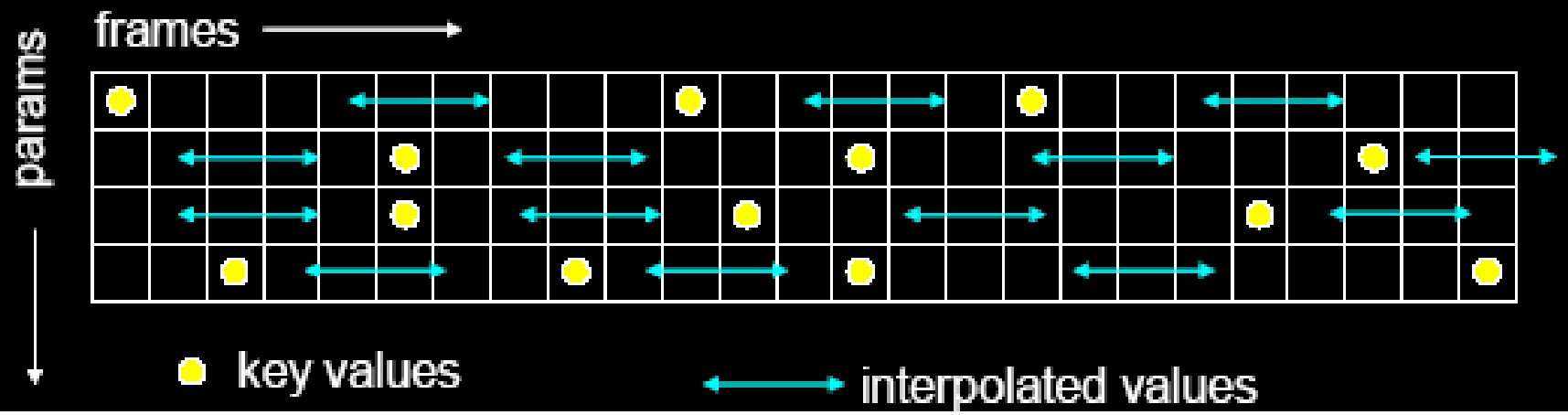
For every vengeance there is an equal and opposite revengeance. This is the one law of animated cartoon motion that also applies to the physical world at large. For that reason, we need the relief of watching it happen to a duck instead.

Cartoon Law Amendment A

A sharp object will always propel a character upward. When poked (usually in the buttocks) with a sharp object (usually a pin), a character will defy gravity by shooting straight up, with

Keyframing Basics

- Despite the name, there aren't really keyframes, *per se*.
- For each variable, specify its value at the "important" frames. Not all variables need agree about which frames are important.
- Hence, *key values* rather than key frames
- Create path for each parameter by interpolating key values



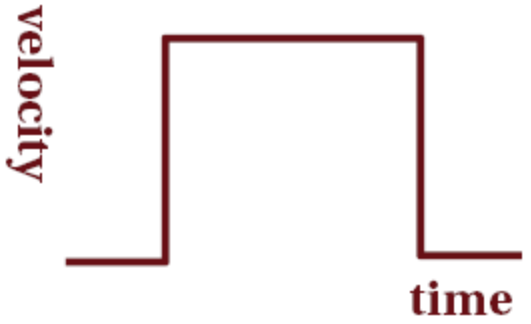
(Pollard http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/slides/25-animII.pdf)

Interpolating Key Frames

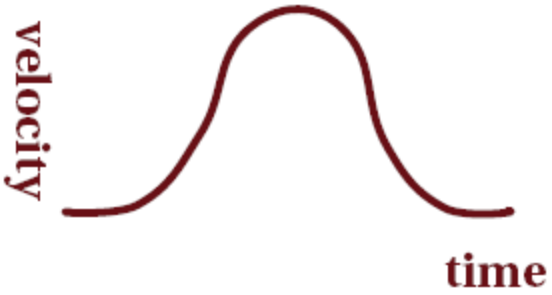
- Can use B-spline/Bezier interpolation curves to interpolate position
- Goals: local control, smooth motion, robustness
- Challenging to maintain the right balance between interpolated position and timing (controlling velocity and acceleration)– almost an art

(Varshney)

Linear



Ease in/ Ease out



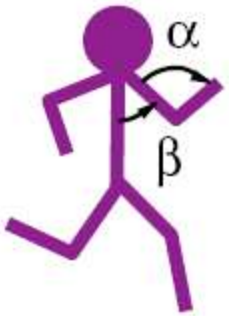
(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Keyframing: Issues

- What should the key values be?
- When should the key values occur?
- How can the key values be specified?
- How are the key values interpolated?
- What kinds of BAD THINGS can occur from interpolation?
  - Invalid configurations (pass through objects)
  - Unnatural motions (painful twists/bends)
  - Jerky motion

(Pollard http://graphics.cs.cmu.edu/nsp/course/15-462/Fall04/slides/25-animII.pdf)

Kinematics -- the study of motion withou regard to the forces that cause it.



Forward: $A = f(\alpha, \beta)$

Inverse: $\alpha, \beta = f^{-1}(A)$

draw graphics

specify fewer degrees of freedom

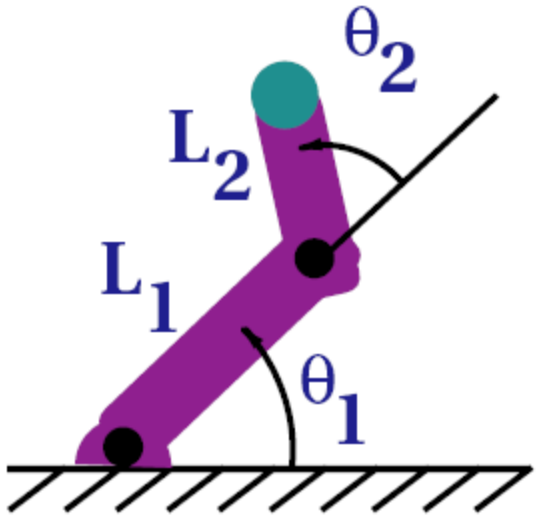
more intuitive control of dof
- pull on hand
- glue feet to the ground

(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Forward Kinematics

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2)$$

$$y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2)$$



$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} = \begin{bmatrix} \text{rot}\,\theta_1 \end{bmatrix} \begin{bmatrix} \text{trans}\,L_1 \end{bmatrix} \begin{bmatrix} \text{rot}\,\theta_2 \end{bmatrix} \begin{bmatrix} \text{trans}\,L_2 \end{bmatrix}$$

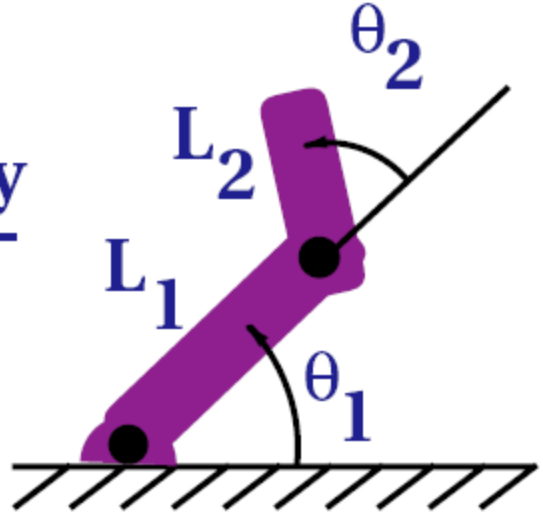
(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Inverse Kinematics

$$\theta_2 = \frac{\cos\ (x^2 + y^2 - L_1^2 - L_2^2)}{2\, L_1\, L_2}$$

$$\theta_1 = \frac{-(L_2 \sin\theta_2)x + (L_1 + L_2 \cos\theta_2)y}{(L_2 \sin\theta_2)y + (L_1 + L_2 \cos\theta_2)x}$$

$$\theta = f^{-1}(x)$$



(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

(Terzopoulos)

Physics-based Animation

- Advantages:
  - Mimics real life more closely
  - Simple to program

- Disadvantages
  - Exact parameters difficult to discern
  - Sometimes *cartoonish* look and feel is preferable to realism

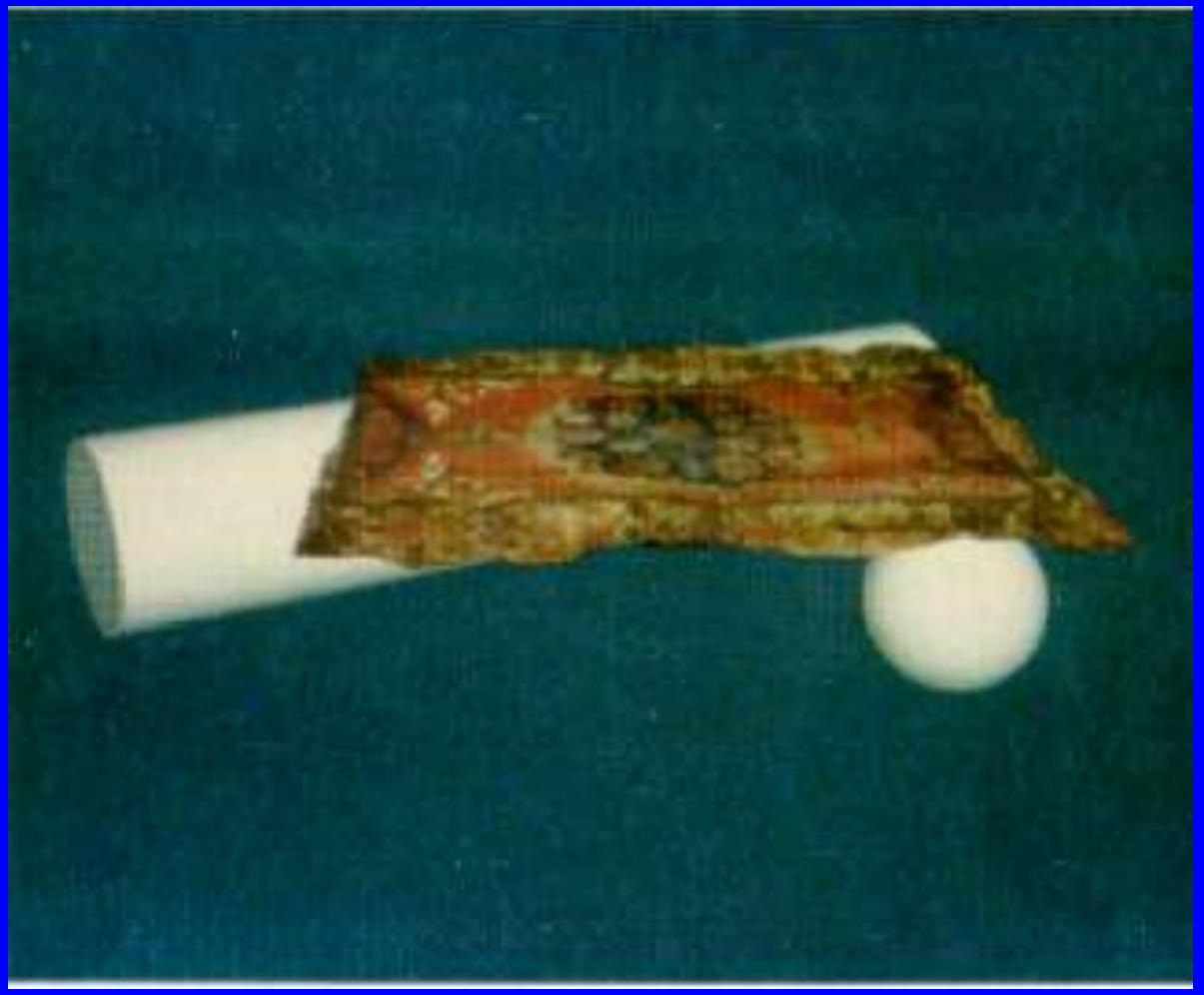
(Varshney)

Physics-based Animation

- Ideally suited for:
  - Large volumes of objects – wind effects, liquids, …
  - Cloth animation/draping
- Underlying mechanisms are usually:
  - Particle systems
  - Mass-spring systems
- Typically solve ordinary or partial differential equations using iterative methods with some initial/ending boundary values and constraints on conservation of mass/energy/angular momentum

Physics-based Animation

- Ideally suited for:
  - Large volumes of objects – wind effects, liquids, …
  - Cloth animation/draping
- Underlying mechanisms are usually:
  - Particle systems
  - Mass-spring systems
- Typically solve ordinary or partial differential equations using iterative methods with some initial/ending boundary values and constraints on conservation of mass/energy/angular momentum



(Terzopoulos, Platt, Barr and Fleischer, SIGGRAGH ’87)

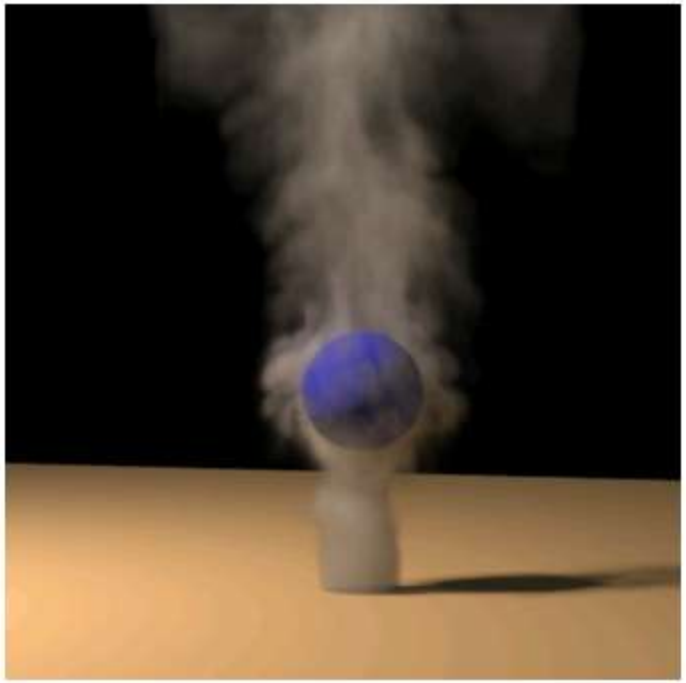
(Terzopoulos, Platt, Barr and Fleischer, SIGGRAGH ’87)

(Terzopoulos, Platt, Barr and Fleischer, SIGGRAGH ’87)

(Terzopoulos, Platt, Barr and Fleischer, SIGGRAGH ’87)

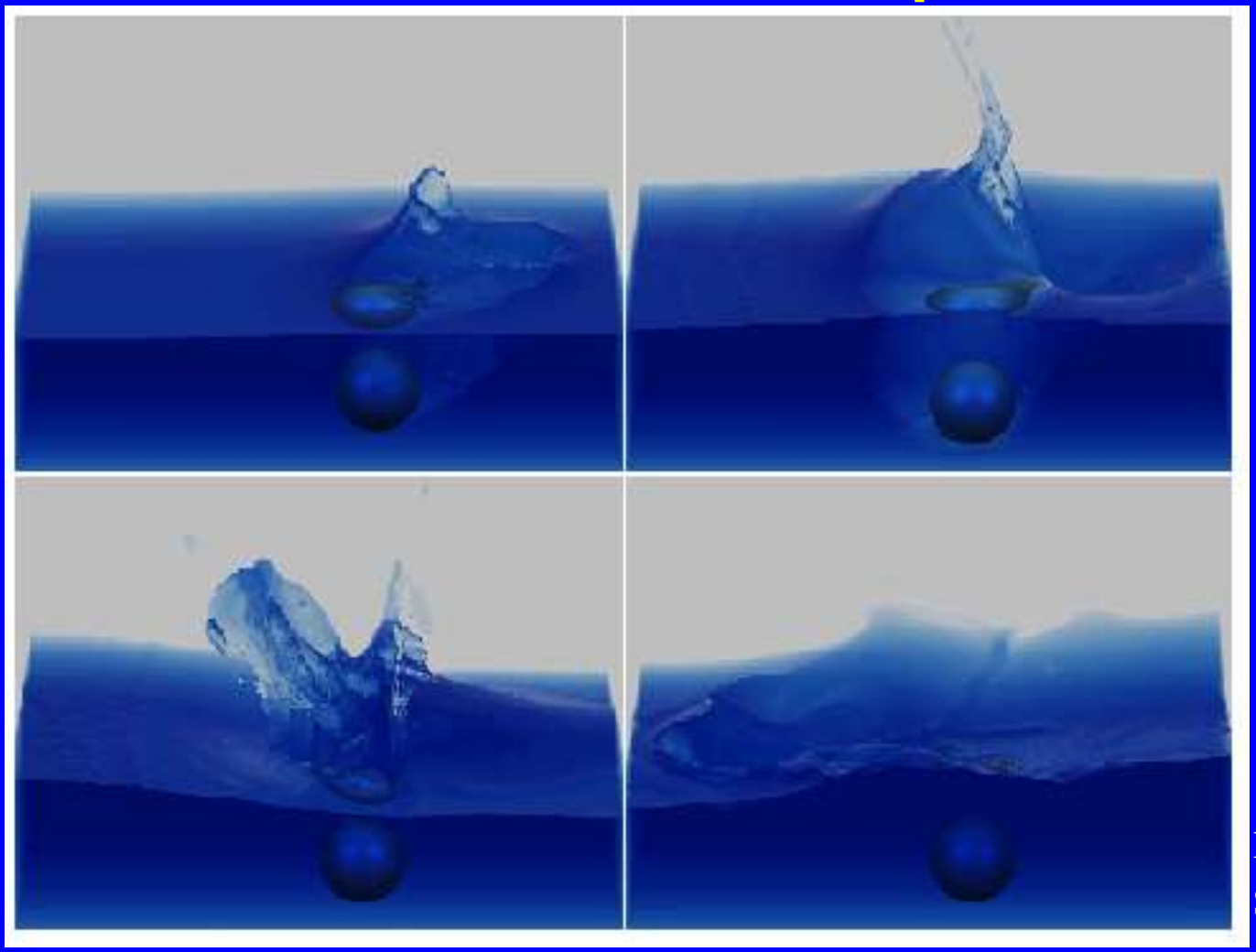
(Terzopoulos, Platt, Barr and Fleischer, SIGGRAGH ’87)

Examples



Images from Fedkiw, Stam, Jensen, SIGGRAPH 2001

Examples



Images from Foster & Fedkiw
SIGGRAPH 2001

Examples

Image courtesy Simon Premoze, Univ. of Utah

Physically real motion

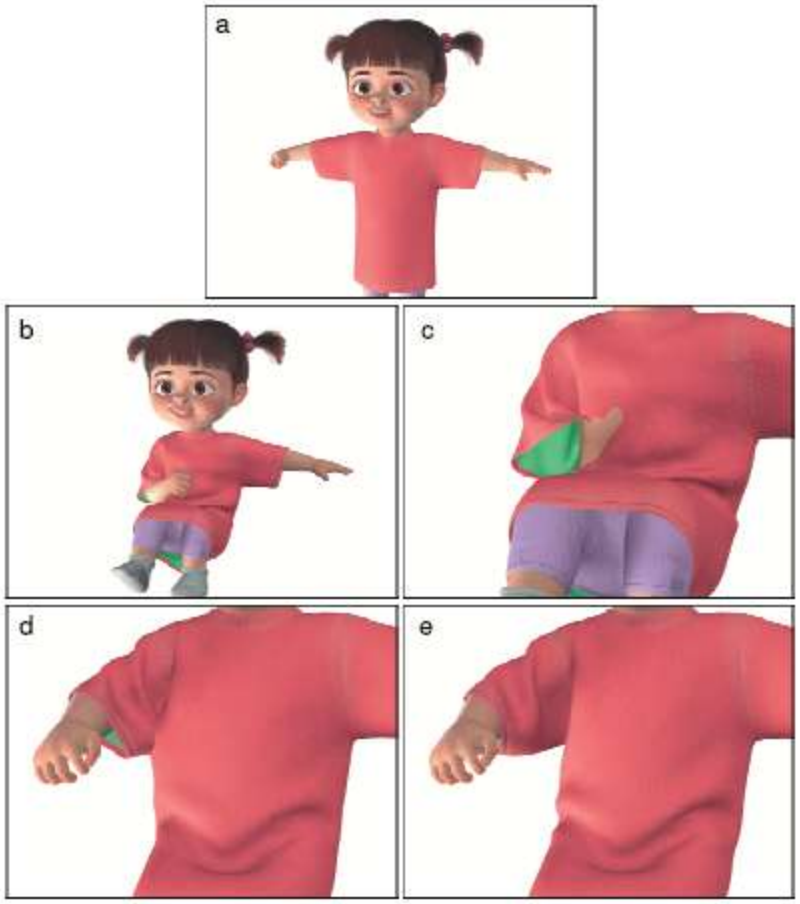


Figure 10: (a) Starting pose. (b) Arm moves in tightly. (c) Close-up view of (b) with right arm invisible. Note how the arm position forces cloth to intersect both itself and the body. (d) Without GIA, a cloth/cloth intersection persists as the arm pulls out, snagging the sleeve. (e) The same frame as (d), but using GIA, the cloth doesn't snag as the arm pulls out.

(http://mrl.nyu.edu/~dt/)

What is Motion Capture?

capture of motion of (human) actor
- whole body
- upper body
- face

more generally...

one way of using a physical device to control animation

- puppeteering
- exoskeletons
- discrete sensors on actors

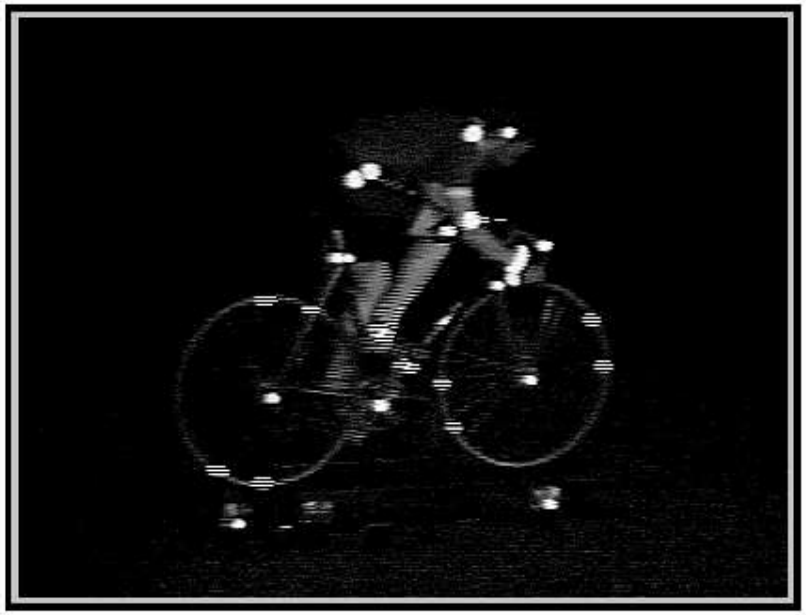
(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Technology––optical

passive reflection––Peak

hand or semi–automatically digitized
time consuming

no glossy or reflective materials
tight clothing
occlusion of markers by props or limbs
higher frames/second

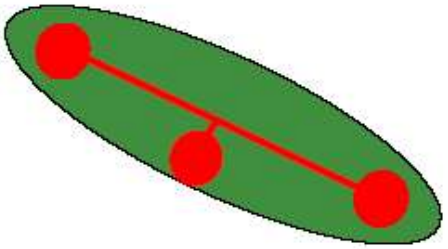


(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Technology--optical

passive reflection--Acclaim, Motion Analysis,...

automatically digitized
240 Hz
not real-time
3 markers/body part for 6 dof
2+ cameras for 3d position data
~$100K



(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Technology––magnetic

electromechanical transducers

Ascension flock of birds

Polhemus Fastrak

limited range/resolution

pigtail (new wireless system)

metal in the environment (treadmill, rebar!)

no identification problem

6 dof information

realtime

low frequency: 30 to 120 Hz

few markers: 10–20

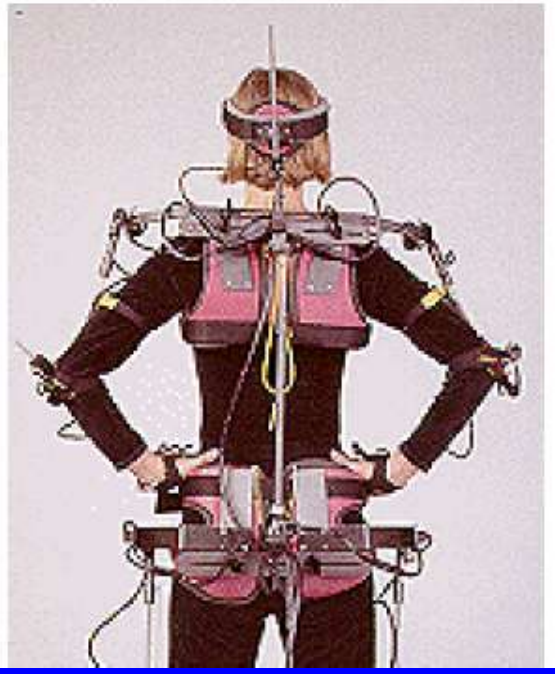
$40K

(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Technology--mechanical

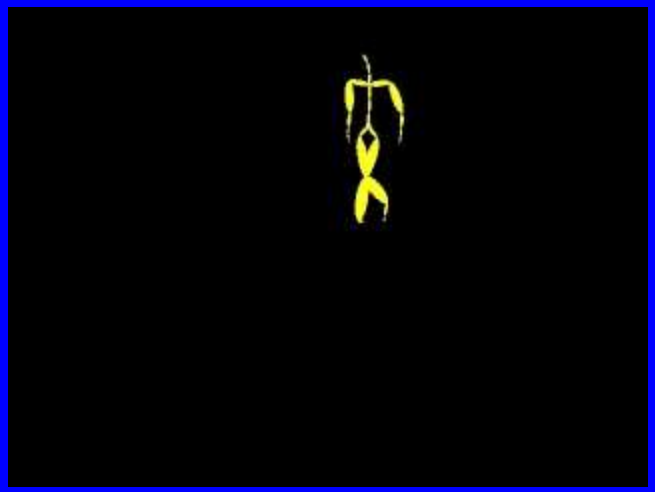
exoskeleton + angle sensors
Analogous

pigtail
no identification problem
realtime
high frequency: 500Hz
not range limited
fit
rigid body approximation



(Hodkins, http://www.cc.gatech.edu/classes/cs8113a_98_spring/)

Motion Capture

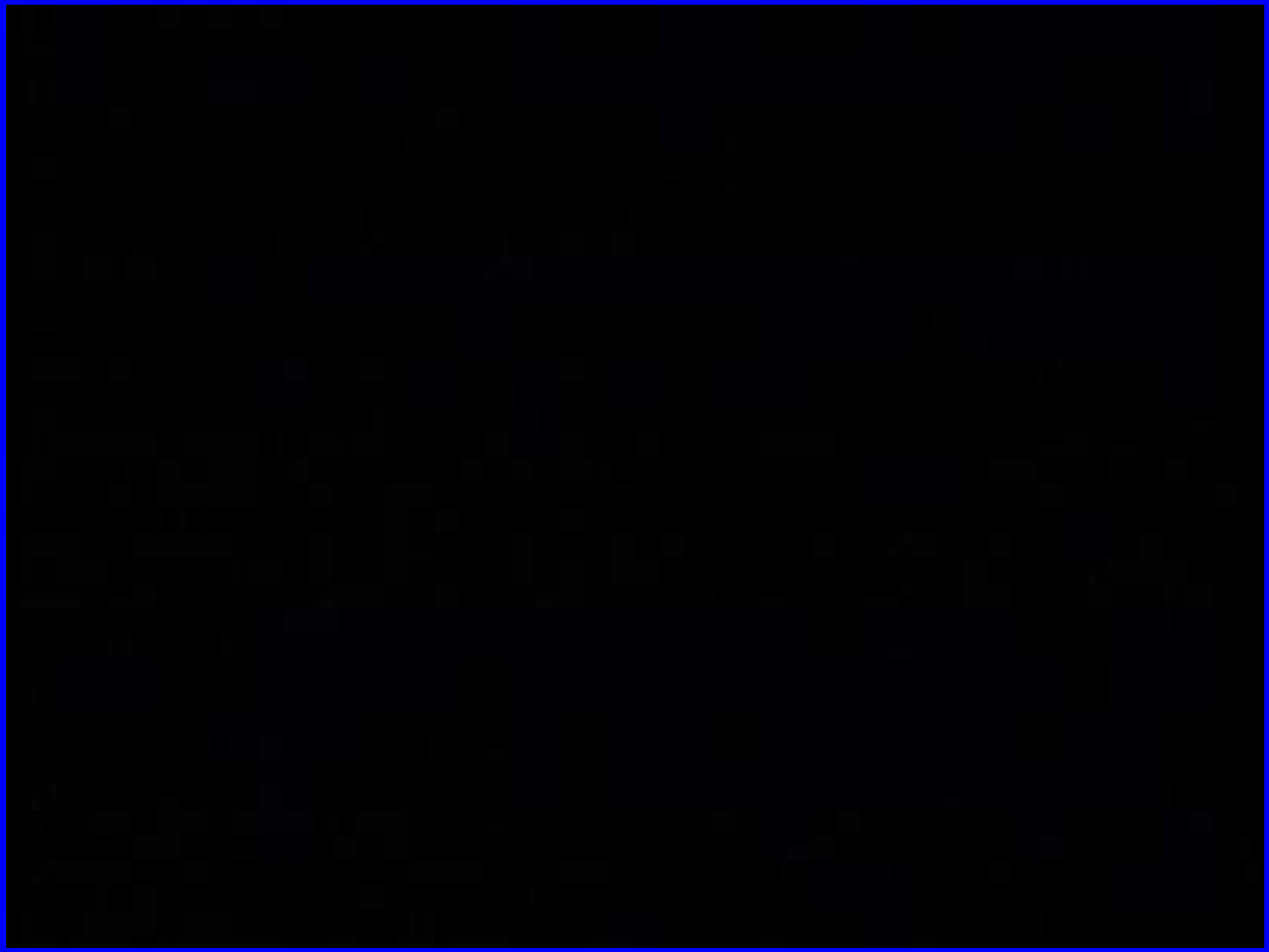


http://mocap.cs.cmu.edu/search.php?subjectnumber=%&motion=%

Behaviors

(Terzopoulos)

(Terzopoulos)



(Terzopoulos)