4. Interpolation-Based Animation

Methods for precisely specifying the motion of objects using basics introduced in previous chapter:

- Key-frame animation systems
- Animation languages
- Object deformation
Key-Frame Systems (good reference: Maya)

In a typical animation system, the key frames are defined & drawn by the master animators, intermediate frames are then drawn by assistant animator.

Relatively simple in computer animation if the shapes are defined by polygons and the correspondence between the vertices is known.
Key-Frame Systems

Order of correspondence is important

Starting Keyframe
Ending Keyframe
Completed Animation
Key-Frame Systems

Order of correspondence is important

Starting frame

ending frame
Key-Frame Systems

For shapes defined by piecewise curves such as composite Bezier curves or B-spline curves, intermediate frames can be generated using surface patch technique:

(i) Interpolating control points of curves in the key frames
Key-Frame Systems

(ii) The path between two adjacent key frames is already known.
(a) the path is a straight line

Construct a surface patch as follows:

\[ S(u, v) = P(u)(1 - b(v)) + b(v) Q(u) \]

where

\[ 0 \leq u, v \leq 1 \]

\[ 0 \leq b(v) \leq 1 \quad b(0) = 0, \quad b(1) = 1 \]

\[ b(v) = v \quad \text{e.g.,} \]

\[ b(v) = 3v^2(1 - v) + v^3 \]
(b) The path is a **cubic curve segment** (Bezier, B-spline)

\[ S(u, v) = C(v) + \{(1 - \alpha(v)[n_1(0) \text{ component of } P(u) - C(0)] \]

\[ + \alpha(v)[n_1(1) \text{ component of } Q(u) - C(1)] \} n_1(v) \]

\[ + \{(1 - \alpha(v)[n_2(0) \text{ component of } P(u) - C(0)] \]

\[ + \alpha(v)[n_2(1) \text{ component of } Q(u) - C(1)] \} n_2(v) \]

\[ + \ldots \]

where \( \alpha(v) \) is a blending function and \( n_1(v) \), \( n_2(v) \) and \( n_3(v) \) are Frenet frame axes at \( C(v) \)
4.2 Animation Languages

What are animation languages?
- *Structured commands* used to encode the information necessary to produce animations

Why do we need animation languages?
- To avoid *overhead* in producing motion sequences
4.2 Animation Languages

Script-based or graphical

- **Script-based**: composed of text instructions, earlier approaches such as ANIMA II, AL,

```plaintext
set position<name><x><y><z>at frame<number>
set rotation<name>[x,y,z]to<angle>at frame<number>
change position<name>to<x><y><z>from frame<number>to frame<number>
change rotation<name>[x,y,z]by<angle>from frame<number>to frame<number>
```
4.2 Animation Languages

**ANIMA II:** released by Reverse System in 2005

**AL:** developed by Steve May at the Advanced Computing Center for the Arts and Design (ACCAD) which is a part of The Ohio State University. Steve is currently employed at Pixar Animation Studios and maintains AL in his spare time.

Download site: http://accad.osu.edu/~smay/AL/download.html
4.2 Animation Languages

- **graphical**: encoding relationships between objects and procedures using acyclic graphs

An animation is represented by a dataflow network such as *Houdini*, *Maya* and most of recent animation languages.
Houdini dataflow network

The object it generates
A collection of nodes connected together that describe the steps needed to accomplish a task is known as a hierarchy of nodes. This hierarchy is created using container nodes.
Advantages of Animation Languages

- Bigger *reusebility, mobility, and alterability*: The script can be used at any time to regenerate it, can be copied and transmitted easily, allows the animation to be iteratively refined (because the script can be incrementally changed and a new animation generated)

- The availability of *programming constructs* allows an algorithmic approach to motion control
4.3 Deforming Objects

- Physically based deformation
  - simulating the flexing of objects undergoing forces

You don’t have much choice, but to follow the physics.
4.3 Deforming Objects

- Physically based deformation

1. Subdividing surface $S(u,v)$
2. Scaling $S(u,v)$ with $T_s$
3. Relocating components that do not change shape $S \circ C_i(t)$
4. Set up feature preserving objective function
5. Set up constraints
6. Performe constrained shape deformation
7. Rendering
4.3 Deforming Objects

- Physically based deformation

The deformation process requires the construction of a feature-preserving objective function. This function is used to determine the shape of the deformed object in an optimization process. Hence, the objective function should be defined as the difference of these two surfaces.

\[ V(u, v) = (\bar{S} - T_s \circ S)(u, v) \]
4.3 Deforming Objects

- Physically based deformation

\[ V(u, v) = (\bar{S} - T_s \circ S)(u, v) \]

The goal is to minimize the energy of the displacement function

\[ E(V) = E_{bending} + E_{gravity} + E_{spring} + E_{moment} + E_{edgeforce} \]

Set to zero
4.3 Deforming Objects

- Physically based deformation

\[ V(u, v) = (\bar{S} - T_s \circ S)(u, v) \]

\[ E_{bending} = \iint_D \kappa \left[ \frac{1}{2} (V_{uu} + V_{vv})^2 - (1 - \sigma)(V_{uu}V_{vv} - V_{uv}^2) \right] dudv \]

\[ E_{stretching} = \frac{1}{2} \iint_D \left[ (2G + \lambda)(V_u^2 + V_v^2) + 2\lambda(V_uV_v) \right] dudv \]

\[ E_{spring} = \frac{1}{2} \iint_D K(\tau_i + \sigma_i)[V(\tau_i, \sigma_i)]^2 dudv \]

Substitute and minimize
4.3 Deforming Objects

- User defined distortion
  - simulator deforms the object directly and defines key shapes
  - usually needs to use non-affine transformations
    - warping
    - coordinate grid deformation
  - an affine transformation is a linear mapping from an affine space to an affine space

Imaging your image is made of rubber and then warp your rubber
Image Warping – non-parametric

- Move control points to specify a spline warp
- Spline produces a smooth vector field
Warp specification - dense

- How can we specify the warp?
  Specify corresponding *spline control points*
  
  - *interpolate* to a complete warping function

But we want to specify only a few points, not a grid.
Warp specification - sparse

- How can we specify the warp?
  Specify corresponding *points*
  - *interpolate* to a complete warping function
  - How do we do it?

How do we go from feature points to pixels?
Triangular Mesh

1. Input correspondences at key feature points
2. Define a triangular mesh over the points
   - Same mesh in both images!
   - Now we have triangle-to-triangle correspondences
3. Warp each triangle separately from source to destination
Warping an Object: picking and pulling

D: displacement vector for seed vertex
n: range of propagation

displace the seed vertex (vertices) and propagate the displacement to adjacent vertices by attenuating initial displacement
Warping an Object: picking and pulling

For a vertex $i$ units away from the seed vertex, the displacement is

$$S(i) D$$

$$S(i) = 1.0 - \left( \frac{i}{n+1} \right)^k$$

$k = 1$: linear attenuation
$k > 1$: create a more elastic impression

CS Dept, UK
Deforming an embedding space (free-form deformation (FFD))

- include the object to be distorted in a local grid and then distort the local grid
- easier to manipulate the local grid than to manipulate the vertices of the object directly
- more powerful than affine transformations because the distortion of the local grid can be nonlinear

\[ f : X \rightarrow Y \]
\[ x \rightarrow Mx + b \]
2D grid deformation

Object to be distorted

Local grid

New vertices are computed using linear interpolation
2D grid deformation

A = (5.0, 12.0)  B = (6.0, 12.0)
C = (5.0, 11.0)  D = (6.0, 11.0)
P = (5.7, 11.6)
2D grid deformation

\[ E = 0.3 \ A' + 0.7 \ B' \]
\[ F = 0.3 \ C' + 0.7 \ D' \]
\[ P' = 0.4 \ F + 0.6 \ E \]
Polyline deformation (2D)

- draw a polyline thru the object to be distorted
- map object vertices to the polyline
- modify the polyline
- map object vertices to the same relative location of the polyline
- suitable for serpentine objects

of the shape of a snake
Polyline deformation (2D)

- draw a polyline thru the object to be distorted
- map object vertices to the polyline
\[ r = \frac{d_2}{d_1} \]

\[ \frac{s_r}{s} = r \]
Free-Form Deformation (FFD): Sederberg
Free-Form Deformation (FFD):

- 3D extension of 2D grid deformation
- superimpose a localized coordinate grid over the object
- register vertices of the object to the grid (local coordinate system)
- manipulate the grid
- map object vertices back into the modified grid, then relocate them in global space
If the local coordinate system is defined by \((\vec{S}, \vec{T}, \vec{U})\) actually \(\vec{S}, \vec{T}\) and \(\vec{U}\) do not have to be unit vectors either.

Relationship between local and global coordinate systems:

\[ P = P_0 + s\vec{S} + t\vec{T} + u\vec{U} \]

\(P, P_0\) : global
\((s, t, u)\) : local

\[ s = ?, \quad t = ?, \quad u = ? \]
\[ P = P_0 + s \vec{S} + t \vec{T} + u \vec{U} \]

\[ P, P_0 : \text{global} \]

\[ (s, t, u): \text{local} \]

\[ s = \left( \vec{T} \otimes \vec{U} \right) \cdot (P - P_0) / \left( \left( \vec{T} \otimes \vec{U} \right) \cdot \vec{S} \right) \]

\[ t = \left( \vec{U} \otimes \vec{S} \right) \cdot (P - P_0) / \left( \left( \vec{U} \otimes \vec{S} \right) \cdot \vec{T} \right) \]

\[ u = \left( \vec{S} \otimes \vec{T} \right) \cdot (P - P_0) / \left( \left( \vec{S} \otimes \vec{T} \right) \cdot \vec{U} \right) \]  

\[ (*) \]

Why?
Here is why:

Since \( \mathbf{T} \otimes \mathbf{U} \perp \mathbf{T} \) and \( \mathbf{T} \otimes \mathbf{U} \perp \mathbf{U} \)
we have

\[
(P - P_0) \cdot (\mathbf{T} \otimes \mathbf{U}) = \left( s \mathbf{S} + t \mathbf{T} + u \mathbf{U} \right) \cdot (\mathbf{T} \otimes \mathbf{U}) = s \mathbf{S} \cdot (\mathbf{T} \otimes \mathbf{U})
\]

Hence

\[
s = \frac{(P - P_0) \cdot (\mathbf{T} \otimes \mathbf{U})}{(\mathbf{S} \cdot (\mathbf{T} \otimes \mathbf{U}))}
\]
**Creation of the localized coordinate grid**

The localized coordinate grid is created using the following formula:

\[ P_{ijk} = P_0 + \frac{i}{l} \vec{S} + \frac{j}{m} \vec{T} + \frac{k}{n} \vec{U} \]

- \( l \): Number of control points in \( \vec{S} \) direction
- \( m \): Number of control points in \( \vec{T} \) direction
- \( n \): Number of control points in \( \vec{U} \) direction
Animator adjusts the locations of the control points $P_{ijk}$ to deform the object.

Deformed position of a vertex of the object is determined through a trivariate Bezier interpolation process:

\[
P(s, t, u) = \sum_{i=0}^{l} \binom{l}{k} (1 - s)^{l-i} s^i .
\]

\[
\left\{ \sum_{j=0}^{m} \binom{m}{j} (1 - t)^{m-j} t^j \left[ \sum_{k=0}^{n} \binom{n}{k} (1 - u)^{n-k} u^k P_{ijk} \right] \right\}
\]

\[0 \leq s, t, u \leq 1\]
The deformation is specified by moving the $P_{i,j,k}$ from their undisplaced, latticial positions. The deformed position $X_{\text{ffd}}$ of an arbitrary point $X$ is found by first computing its $(s, t, u)$ coordinates from equation (*), and then evaluating the vector valued tri-variate Bernstein polynomial:

$$X_{\text{ffd}} = \sum_{i=0}^{l} \binom{l}{k} (1-s)^{l-i} s^i.$$ 

$$\left\{ \sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^j \left[ \sum_{k=0}^{n} \binom{n}{k} (1-u)^{n-k} u^k P_{ijk} \right] \right\}$$
FFDs can be composed
- sequentially vs hierarchically

Detail elements can be added in stages
usually in the same direction

Allows the user to work at various levels of detail

can be sub-dimensional or multi-dimensional
Free form deformation through control of parametric surfaces
Free form deformation through control of parametric surfaces
End of Interpolation IV