Shape Interpolation Algorithm

- To compute a blended vertex position:
  \[ \mathbf{v}' = \mathbf{v}_{base} + \sum \phi_i \cdot (\mathbf{v}_i - \mathbf{v}_{base}) \]

- The blended position is the base position plus a contribution from each target whose DOF value is greater than 0

- To blend the normals, we use a similar equation:
  \[ \mathbf{n}' = \mathbf{n}_{base} + \sum \phi_i \cdot (\mathbf{n}_i - \mathbf{n}_{base}) \]

- We won’t normalize them now, as that will happen later in the skinning phase
Smooth Skin Algorithm

The deformed vertex position is a weighted average over all of the joints that the vertex is attached to. Each attached joint transforms the vertex as if it were rigidly attached. Then these values are blended using the weights:

\[ \mathbf{v}' = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{v}' \]

Where:
- \( \mathbf{v}' \) is the final vertex position in world space
- \( w_i \) is the weight of joint \( i \)
- \( \mathbf{v}' \) is the untransformed vertex position (output from the shape interpolation)
- \( \mathbf{B}_i \) is the binding matrix (world matrix of joint \( i \) when the skin was initially attached)
- \( \mathbf{W}_i \) is the current world matrix of joint \( i \) after running the skeleton forward kinematics

Note:
- \( \mathbf{B} \) remains constant, so \( \mathbf{B}^{-1} \) can be computed at load time
- \( \mathbf{B}^{-1} \cdot \mathbf{W} \) can be computed for each joint before skinning starts
- All of the weights must add up to 1:

\[ \sum w_i = 1 \]
Layered Approach

- We use a simple layered approach
  - Skeleton Kinematics
  - Shape Interpolation
  - Smooth Skinning
- Most character rigging systems are based on some sort of layered system approach combined with general purpose data flow to allow for customization
Free-Form Deformation
Global Deformations

- A global deformation takes a point in 3D space and outputs a deformed point

\[ x' = F(x) \]

- A global deformation is essentially a deformation of space

- Smooth skinning is technically not a global deformation, as the same position in the initial space could end up transforming to different locations in deformed space
Free-Form Deformations

- FFDs are a class of deformations where a low detail control mesh is used to deform a higher detail skin.
- Generally, FFDs are classified as global deformations, as they describe a mapping into a deformed space.
- There are a lot of variations on FFDs based on the topology of the control mesh.
Lattice FFDs

- The original type of FFD uses a simple regular lattice placed around a region of space
- The lattice is divided up into a regular grid (4x4x4 points for a cubic deformation)
- When the lattice points are then moved, they describe smooth deformation in their vicinity
Lattice FFDs

- We start by defining the undeformed lattice space:

\[
x(s, t, u) = x_0 + s \cdot s_0 + t \cdot t_0 + u \cdot u_0
\]

\[
0 \leq s \leq 1 \quad 0 \leq t \leq 1 \quad 0 \leq u \leq 1
\]

\[
M = \begin{bmatrix}
    s_x & s_y & s_z & 0 \\
    t_x & t_y & t_z & 0 \\
    u_x & u_y & u_z & 0 \\
    x_{0x} & x_{0y} & x_{0z} & 1
\end{bmatrix}
\]
Lattice FFDs

- We then define the number of sections in the 3 lattice dimensions:

\[ 1 \leq l, m, n \leq 3 \]

- And then set the initial lattice positions: \( p_{ijk} \)

\[
p_{ijk} = x_0 + \frac{i}{l} \cdot s_0 + \frac{j}{m} \cdot t_0 + \frac{k}{n} \cdot u_0
\]
Lattice FFDs

- To deform a point $x$, we first find the $(s,t,u)$ coordinates:

$$\begin{bmatrix} s & t & u & 1 \end{bmatrix} = x \cdot M^{-1}$$

- Then deform that into world space:

$$x_{ffd} = \sum_{i=0}^{l} \binom{l}{i} (1 - s)^{-i} s^i \cdot \left( \sum_{j=0}^{m} \binom{m}{j} (1 - t)^{m-j} t^j \cdot \left( \sum_{k=0}^{n} \binom{n}{k} (1 - u)^{n-k} u^k \cdot p_{ijk} \right) \right)$$
Arbitrary Topology FFDs

- The concept of FFDs was later extended to allow an arbitrary topology control volume to be used.
Axial Deformations & WIRES

- Another type of deformation allows the user to place lines or curves within a skin.
- When the lines or curves are moved, they distort the space around them.
- Multiple lines & curves can be placed near each other and will properly interact.
Surface Oriented FFDs

- This modern method allows a low detail polygonal mesh to be built near the high detail skin.
- Movement of the low detail mesh deforms space nearby.
- This method is nice, as it gives a similar type of control that one gets from high order surfaces (subdivision surfaces & NURBS) without any topological constraints.
Surface Oriented FFDs
Using FFDs

- FFDs provide a high level control for deforming detailed geometry
- Still, we must address the issue of how to animate and deform the FFD mesh
- The verts in the mesh can be animated with the smooth skinning algorithm, shape interpolation, or other methods
Body Scanning
Body Scanning

- Data input has become an important issue for the various types of data used in computer graphics
- Examples:
  - Geometry: Laser scanners
  - Motion: Optical motion capture
  - Materials: Gonioreflectometer
  - Faces: Computer vision
- Recently, people have been researching techniques for directly scanning human bodies and skin deformations
Body Scanning

- Practical approaches tend to use either a 3D model scanner (like a laser) or a 2D image based approach (computer vision).
- The skin is scanned at various key poses and some sort of 3D model is constructed.
- Some techniques attempt to fit this back onto a standardized mesh, so that all poses share the same topology. This is difficult, but it makes the interpolation process much easier.
- Other techniques interpolate between different topologies. This is difficult also.
Body Scanning
Body Scanning
Anatomical Modeling
Anatomical Modeling

- The motion of the skin is based on the motion of the underlying muscle and bones. Therefore, in an anatomical simulation, the tissue beneath the skin must be accounted for.
- One can model the bones, muscle, and skin tissue as deformable bodies and then use physical simulation to compute their motion.
- Various approaches exist ranging from simple approximations using basic primitives to detailed anatomical simulations.
Skin & Muscle Simulation

- Bones are essentially rigid
- Muscles occupy almost all of the space between bone & skin
- Although they can change shape, muscles have essentially constant volume
- The rest of the space between the bone & skin is filled with fat & connective tissues
- Skin is connected to fatty tissue and can usually slide freely over muscle
- Skin is anisotropic as wrinkles tend to have specific orientations
Simple Anatomical Models

- Some simplified anatomical models use ellipsoids to model bones and muscles
Simple Anatomical Models

- Muscles are attached to bones, sometimes with tendons as well
- The muscles contract in a volume preserving way, thus getting wider as they get shorter
Simple Anatomical Models

- Complex musculature can be built up from lots of simple primitives
Simple Anatomical Models

- Skin can be attached to the muscles with springs/dampers and physically simulated with collisions against bone & muscle.
Detailed Anatomical Models

- One can also do detailed simulations that accurately model bone & muscle geometry, as well as physical properties.
- This is becoming an increasing popular approach, but requires extensive set up.
- Check out cgcharacter.com
Detailed Anatomical Models
Pose Space Deformation
Pose Space Deformation

“Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation”

J. P. Lewis, Matt Cordner, Nickson Fong
Paper Outline

1. Introduction
2. Background
3. Deformation as Scattered Interpolation
4. Pose Space Deformation
5. Applications and Discussion
6. Conclusion
Key Goals of a Skinning System

- “The algorithm should handle the general problem of skeleton-influenced deformation rather than treating each area of anatomy as a special case. New creature topologies should be accommodated without programming or considerable setup efforts.”
Key Goals of a Skinning System

- “It should be possible to specify arbitrary desired deformations at arbitrary points in the parameter space, with smooth interpolation of the deformation between these points.”
Key Goals of a Skinning System

- “The system should allow direct manipulation of the desired deformations”
Key Goals of a Skinning System

- “The locality of deformation should be controllable, both spatially and in the skeleton’s configuration space (pose space).”
Key Goals of a Skinning System

“In addition, we target a conventional animator-controlled work process rather than an approach based on automatic simulation. As such we require that animators be able to visualize the interaction of a reasonably high-resolution model with an environment in real time. Real time synthesis is also required for applications such as avatars and computer games”
Paper Outline (section 2)

- 2. Background
  - 2.1 Surface Deformation Models
  - 2.2 Multi-Layered and Physically Inspired Models
  - 2.3 Common Practice
    - 2.3.1 Shape Interpolation
    - 2.3.2 Skeleton-Subspace Deformation
    - 2.3.3 Unified Approaches
- 2.4 Kinematic or Physical Simulation?
Key Technology

- Scattered Data Interpolation Using Radial Basis Functions
Key Technology

- Scattered Data Interpolation Using Radial Basis Functions
- Huh?
Interpolation

- Interpolation vs. Extrapolation
- Linear Interpolation vs. Higher Order
- Structured vs. Scattered
- 1-Dimensional vs. Multi-Dimensional
- Interpolation vs. Approximation
Interpolation Techniques

- Splines (cubic, B-splines, NURBS…)
- Series (polynomial, Fourier, radial basis functions, wavelets…)
- Rational functions
- Exact solution, minimization, fitting, approximation
Radial Basis Functions

- What is a radial basis function?
- How do we use them to interpolate data?
What is an RBF?

- A radial basis function (RBF) is simply a function based on a scalar radius: 
  \( _{(r)} \)
- We can use it as a spherically symmetric function based on the distance from a point
- In 3D space, for example, you can think of a field emanating from a point that is symmetric in every direction (like a gravitational field of a planet)
- The value of that field is based entirely on the distance from the point (i.e., the radius)
Radial Basis Functions

- If we placed a RBF at location $\mathbf{x}_k$ in space, and we want to know the value of the field at location $\mathbf{x}$, we just compute:

$$\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(|\mathbf{x}-\mathbf{x}_k|)$$

- This works with an $\mathbf{x}$ of any number of dimensions
Radial Basis Functions

- What function should we use for \( _\psi(r) \)?
- Well, technically, we could use any function we want
- We will choose to use a Gaussian:

\[
\psi(r) = \exp \left( \frac{-r^2}{2\sigma^2} \right)
\]
Why use a Gaussian RBF?
- We want a function that has a localized influence that drops off to 0 at a distance
- We want to be able to adjust the range of influence (that’s what _ is for)
- We want a smooth function
- We want a function whose value is 1 at r=0
- We want the first derivative to be 0 at r=0. This causes the function to be flat across the top and avoids spikes
- We want to use something that is relatively fast to compute
How Do We Use RBFs?

- How do we use radial basis functions to interpolate scattered data?
- We define the interpolated value at a point as a weighted sum of radial basis functions:

\[ \hat{d}(x) = \sum_{k=1}^{N} w_k \psi \left( \frac{|x - x_k|}{\sigma_k} \right) \]

- The RBFs must be positioned and the weights adjusted so that the result best approximates the scattered data and smoothly interpolates the space between the data points.
Example in One Dimension

\[ d_1 = w_1 + w_2 \psi \left( |x_2 - x_1| \right) + w_3 \psi \left( |x_3 - x_1| \right) \]
Solving For Weight Values

\[ d_1 = w_1 + w_2 \psi \left( x_2 - x_1 \right) + w_3 \psi \left( x_3 - x_1 \right) \]
\[ d_2 = w_1 \psi \left( x_1 - x_2 \right) + w_2 + w_3 \psi \left( x_3 - x_2 \right) \]
\[ d_3 = w_1 \psi \left( x_1 - x_3 \right) + w_2 \psi \left( x_2 - x_3 \right) + w_3 \]
Solving For Weight Values

\[ d_1 = w_1 + w_2 \psi \left( x_2 - x_1 \right) + w_3 \psi \left( x_3 - x_1 \right) \]
\[ d_2 = w_1 \psi \left( x_1 - x_2 \right) + w_2 + w_3 \psi \left( x_3 - x_2 \right) \]
\[ d_3 = w_1 \psi \left( x_1 - x_3 \right) + w_2 \psi \left( x_2 - x_3 \right) + w_3 \]

\[
\begin{bmatrix}
  d_1 \\
  d_2 \\
  d_3
\end{bmatrix} =
\begin{bmatrix}
  1 & \psi_{21} & \psi_{31} \\
  \psi_{12} & 1 & \psi_{32} \\
  \psi_{13} & \psi_{23} & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
  w_1 \\
  w_2 \\
  w_3
\end{bmatrix}
\]
Solving For Weight Values

\[
\begin{align*}
d_1 &= w_1 + w_2 \psi \left( x_2 - x_1 \right) + w_3 \psi \left( x_3 - x_1 \right) \\
d_2 &= w_1 \psi \left( x_1 - x_2 \right) + w_2 + w_3 \psi \left( x_3 - x_2 \right) \\
d_3 &= w_1 \psi \left( x_1 - x_3 \right) + w_2 \psi \left( x_2 - x_3 \right) + w_3
\end{align*}
\]

\[
\begin{bmatrix}
d_1 \\
d_2 \\
d_3
\end{bmatrix} =
\begin{bmatrix}
1 & \psi_{21} & \psi_{31} \\
\psi_{12} & 1 & \psi_{32} \\
\psi_{13} & \psi_{23} & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
w_1 \\
w_2 \\
w_3
\end{bmatrix}
\]

\[\mathbf{d} = \mathbf{\Phi} \cdot \mathbf{w}\]
Solving For Weight Values

\[
\begin{align*}
    d_1 &= w_1 + w_2 \psi \left( x_2 - x_1 \right) + w_3 \psi \left( x_3 - x_1 \right) \\
    d_2 &= w_1 \psi \left( x_1 - x_2 \right) + w_2 + w_3 \psi \left( x_3 - x_2 \right) \\
    d_3 &= w_1 \psi \left( x_1 - x_3 \right) + w_2 \psi \left( x_2 - x_3 \right) + w_3
\end{align*}
\]

\[
\begin{bmatrix}
    d_1 \\
    d_2 \\
    d_3
\end{bmatrix} =
\begin{bmatrix}
    1 & \psi_{21} & \psi_{31} \\
    \psi_{12} & 1 & \psi_{32} \\
    \psi_{13} & \psi_{23} & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
    w_1 \\
    w_2 \\
    w_3
\end{bmatrix}
\]

\[
d = \varnothing \cdot w
\]

\[
w = \varnothing^{-1} \cdot d
\]
Inverse vs. Pseudoinverse

- If the matrix is square, you can use the inverse
  \[ w = \mathbf{\Theta}^{-1} \cdot \mathbf{d} \]

- Otherwise, you can use the pseudoinverse, which has similar properties, but works for non-square matrices:
  \[ w = \left( \mathbf{\Theta}^T \mathbf{\Theta} \right)^{-1} \mathbf{\Theta}^T \mathbf{d} \]
PSD (pose-space deformation) uses SDI (scattered data interpolation) as an improved technique for shape interpolation.

As RBFs (radial basis functions) drop to 0 away from the data points, it’s nice if you use them to interpolate functions that are close to 0. Therefore, they subtract off the default pose and treat all other poses as deviations from the default pose.

They describe several other details of the implementation in sections 4 & 5.
Performance of PSD

- At runtime, to compute a deformed vertex position, one must evaluate:

\[ \hat{d}(x) = \sum_{k} w_k \psi \left( |x - x_k| \right) \]

for each component of the vertex.

- We can expand this to:

\[ v' = v_{base} + \sum_{i} w_i \exp \left( \frac{|\bar{\bar{\Omega}} - \bar{\Omega}_i|^2}{2\sigma^2} \right) \]
Performance of PSD

- Compare to simple morphing:

\[ v' = v_{base} + \sum_{i=1}^{M} \phi_i \cdot (v_i - v_{base}) \]

- PSD:

\[ v' = v_{base} + \sum_{i=1}^{N} w_i \exp \left( \frac{|\bar{\Omega} - \bar{\Omega}_i|^2}{2\sigma^2} \right) \]
Memory Usage of PSD

- With morphing, every vertex must store $M \times 3$ floats, where $M$ is the number of targets that affect the vertex.
- With PSD, every vertex must store $N \times 3 + N \times R$ floats, where $N$ is the number of poses for the vertex and $R$ is the number of DOFs affecting the vertex.
Surface Oriented Free Form Deformations
Surface Oriented Free Form Deformation

- “Skinning Characters using Surface-Oriented Free-Form Deformations”
- Karan Singh
- Evangelos Kokkevis
Paper Outline

1. Introduction
2. Free-Form Deformation Techniques
3. Surface-Oriented Deformations
   - 3.1 Overview of the Algorithm
   - 3.2 Registration
   - 3.3 Deformation
4. Algorithm Analysis
5. Skinning Workflow
6. Results and Conclusion
Registration Phase

- When the model is set up, every vertex in the high detail mesh must be attached to nearby triangles in the low detail mesh.
- The attachment weights are based on a distance function:
  \[ f(d, \text{local}) = \frac{1}{1 + d^{\text{local}}} \]
- And then normalized (so they sum up to 1).
- A vertex will generally only attach to a small number of triangles.
- For every attachment, we find the coordinates in the triangle’s space.
Registration

- To find the vertex position relative to the control triangle i, we build a registration matrix $R_i$ that defines the triangle’s space

\[
\begin{align*}
    a &= t_2 - t_1 \\
    b &= t_3 - t_1 \\
    c &= \frac{a \times b}{|a \times b|} \\
    d &= t_1
\end{align*}
\]

- Note: I use different notation than the paper
Registration

- Both the high detail skin and the low detail control mesh are constructed in the skin local space.
- If a vertex on the high detail skin is \( \mathbf{v} \), then its position \( \mathbf{v}^* \) in triangle \( i \)'s space is:

\[
\mathbf{v}^* = \mathbf{R}_i^{-1} \cdot \mathbf{v}
\]
Deformation

- When the triangles of the control mesh get positioned into world space, we compute a new deformation matrix $D$ using the same technique as we used to compute $R$.
- Then, we transform the triangle-local vertex by this matrix.
- This is done for all triangles affecting a vertex, and then we take a weighted sum

$$v' = \sum w_i D_i \cdot R_i^{-1} \cdot v$$
Deformation

- This looks familiar...

\[ v' = \sum w_i D_i \cdot R_i^{-1} \cdot v \]
Deformation

- This looks familiar…

\[ v' = \sum w_i D_i \cdot R_i^{-1} \cdot v \]

- When compared to:

\[ v'' = \sum w_i W_i \cdot B_i^{-1} \cdot v' \]

- We see that SOFFD’s do the exact same math as smooth skinning!

- Instead of using matrices from skeletal joints they use matrices based on triangles
Layered Approach

- Skeleton Posing
- Pose Space Deformation
- Surface Oriented Free Form Deformation
- High Order Surface Tessellation