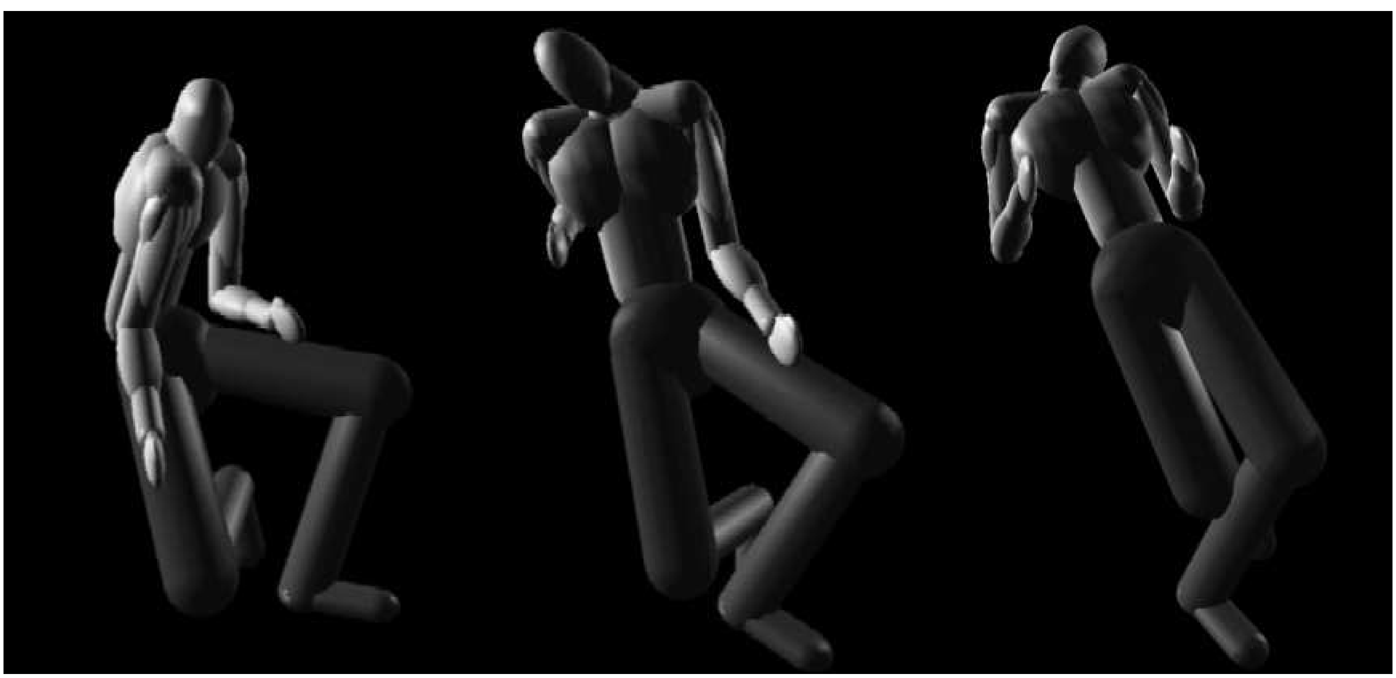
Project #5

Keyframe Animation

Main Goal:

- Load in a hierarchical model of an actor:
  - Joints (variable transforms, DOFs), offsets (rigid translations), geometry (.ray files)
- Load in a sequence of keyframe data
  - Match the DOFs to the variable transform matrices
- Animate the hierarchical model.
  - Compute the interpolated DOFs as a function of wall clock time.
  - Update the model’s transform matrices
  - Redraw the model.
  - Repeat for every redraw.



main.cpp

- Parses the command line and invokes KeyRay()

raydata, cone, cube, … (.h)(.cpp)

- Identical to Project #3
- We use these data structures to describe the scene graph of the hierarchical model

parser(.h)(.cpp)

- Identical to Project #3
- ParseRayFile() invoked by the actor file parser to load the .ray files describing its parts

bmp(.h)(.cpp)

- Identical to Project #3
- Used by the ray file parser to load textures
- You can use this to save your output to make movies

keyray(.h)(.cpp)

- The main code of the key frame animator
- Contains the given code for:
  - The GLUT OpenGL renderer and GUI
  - The actor file (.act) file parser, ParseActFile()
  - An interface to the SVD fitting code, FitSpline()
  - A call to find the wall clock time, GetTime()
- Contains your code to:
  - Parse key frame data files, ParseKeyFile()
  - Update the model in time, Idle()
  - Fit splines to the key frame data
  - Update model transform matrices from the DOFs
  - Your features…

SVD, SVDFit, MatrixMNTC(.h)(.cpp)

- Code for performing SVD to compute a least squares fit. (i.e. Ax=b, x=?)

*.act, *.ray, *.key

- Two actor (.act) files and their geometry (.ray's)
- Several sets of key frame data (.key)

File Formats

Actor files (.act):

```
#DOFS 44
 #JOINT ROOT_NATURAL--XT TranslateX
  #OFFSET 0 0 0
  #RAYFILE NONE
  #CHILDREN 1
   #JOINT ROOT_NATURAL--YT TranslateY
    #OFFSET 0 0 0
    #RAYFILE NONE
    #CHILDREN 1
     #JOINT ROOT_NATURAL--ZT TranslateZ
      #OFFSET 0 0 0
      #RAYFILE NONE
      #CHILDREN 1
       #JOINT ROOT_NATURAL--YR RotateY
        #OFFSET 0 0 0
        #RAYFILE NONE
        #CHILDREN 1
         #JOINT ROOT_NATURAL--ZR RotateZ
          #OFFSET 0 0 0
          #RAYFILE NONE
          #CHILDREN 1
           #JOINT ROOT_NATURAL--XR RotateX
            #OFFSET 0 0 0
            #RAYFILE muscle.pelvis.ray
            #CHILDREN 3
                 …
```

#DOFS n - Number of degrees of freedom = n

#JOINT dof_name dof_type – The DOF variable name and one of the six types of transformations (*or NoMotion – not a DOF!*).

#OFFSET x y z – A static translation *before* this joint and but *after* the parent joint.

#RAYFILE fname.ray – A ray file containing the geometry to be drawn *after* this joint.

#CHILDREN n – Number of child joints = n

File Formats (cont'd)

Keyframe data files (.key):

```
#DOFS 44 ROOT_NATURAL--XT ROOT_NATURAL--YT ROOT_NATURAL—ZT
          ROOT_NATURAL--YR ROOT_NATURAL--ZR ROOT_NATURAL--XR . . .
#DURATION 3.5332980
#FRAMES 107
 0.0       0.8343309 0.0        0.0       -0.0453087 0.1133637 . . .
-0.0495591 0.8284861 0.0931351 -0.3131819 -0.0631809 0.1525526 . . .
 . . .
```

#DOFS d s1 s2 s3 … sd - Number of degrees of freedom, d, followed by their d names. Listed *in order* of appearance in the data.

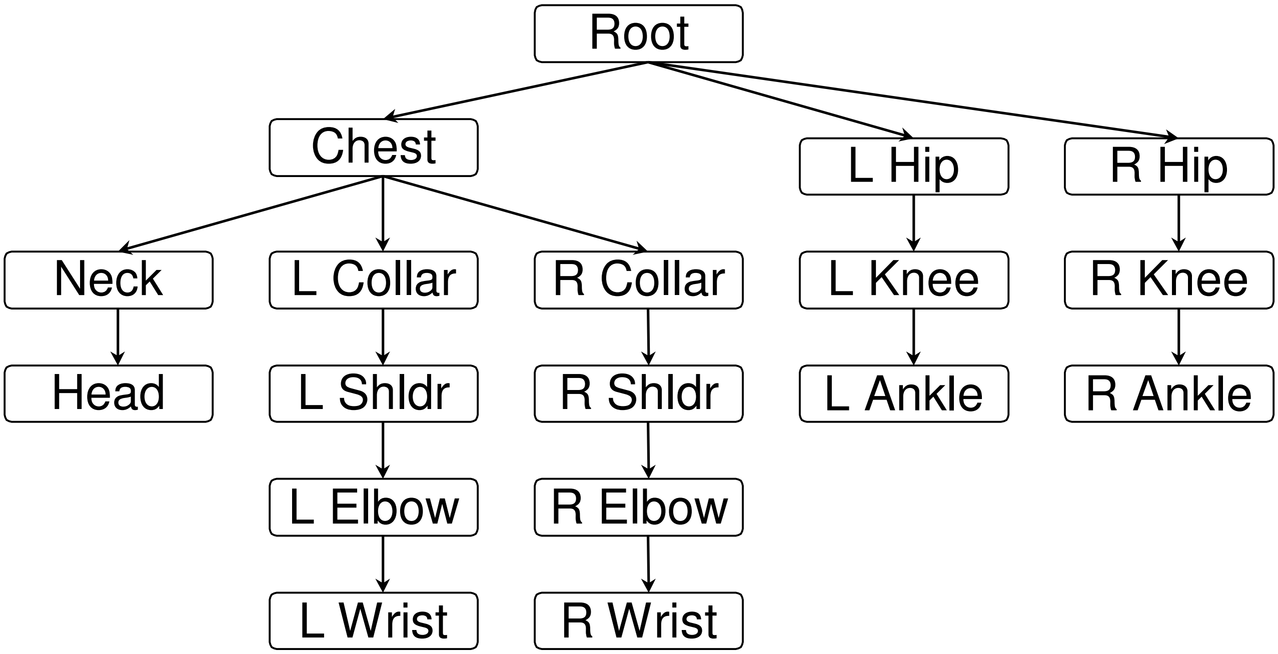
#DURATION t **–** The time, t, in seconds, which the key frame data spans.

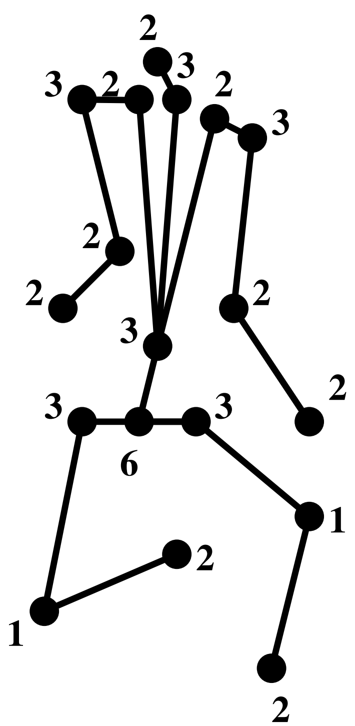
#FRAMES k – The number, k, of key frames. Followed by k lines of key frame data (*equally* spaced in time), such that, if the i^{th} DOF's value in the j^{th} keyframe is f(i,j), then

$$
\begin{array}{l}
f(1,1)\; f(2,1)\; f(3,1)\; f(4,1)\; f(5,1)\; f(6,1)\; \ldots \\
f(1,2)\; f(2,2)\; f(3,2)\; f(4,2)\; f(5,2)\; f(6,2)\; \ldots \\
f(1,2)\; f(2,3)\; f(3,3)\; f(4,3)\; f(5,3)\; f(6,3)\; \ldots \\
\ldots
\end{array}
$$

and keyframe j=1 corresponds to time=0 while keyframe j=k corresponds to time=t (the duration).

Our Actor


Root
Chest
L Hip
R Hip
Neck
L Collar
R Collar
L Knee
R Knee
Head
L Shldr
R Shldr
L Ankle
R Ankle
L Elbow
R Elbow
L Wrist
R Wrist



What you need to do:

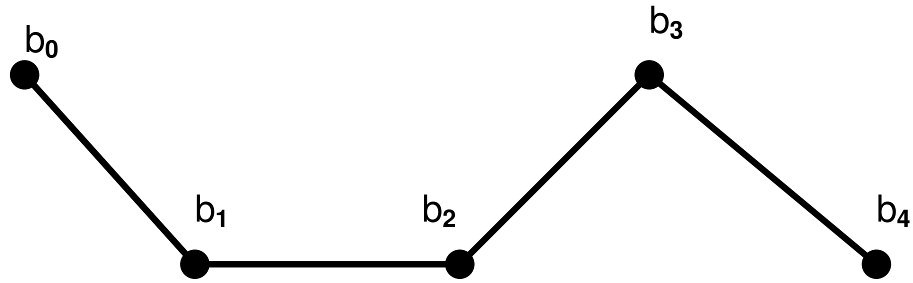
- Develop some data structures to parse the keyframe data into
- Implement a parser to load in the keyframe data
- Create functions to interpolate the keyframe data at arbitrary times (linear interpolations and evaluation of fitted splines)
- Fill out the GLUT Idle() callback to fetch the wall clock time, evaluate the interpolated DOF values, and update the scene graph matrices
- glutPostRedisplay()
- Optional features…

DOF Data Structure:

- DOF name (e.g. LeftHip—XR)
- DOF type (e.g. RotateX, TranslateX)
- Keyframe total time duration
- Keyframe data elements
- Pointer to the TreeNode whose matrix depends upon this DOF. GetDof(char *dof_name) will return this pointer.
- Uniform B-Spline control points.
- Other things appropriate to your implementation.

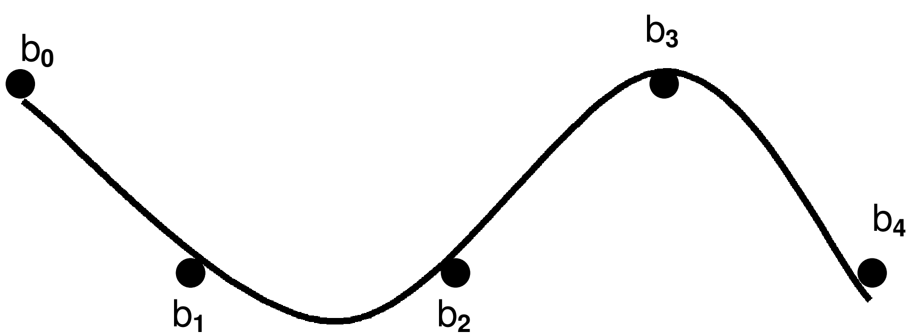
Keyframe Data Interpolation

Linear Interpolation:



- At times between the keyframe DOF values, b_i, linearly interpolate between the two closest keyframe values.

Spline Fitting:



- At each time evaluate a uniform cubic B-Spline whose control points (not shown) are chosen to yield a fit to the keyframe data with least squares error.

Uniform Cubic B-Splines:

- See class notes for details.

Overview:

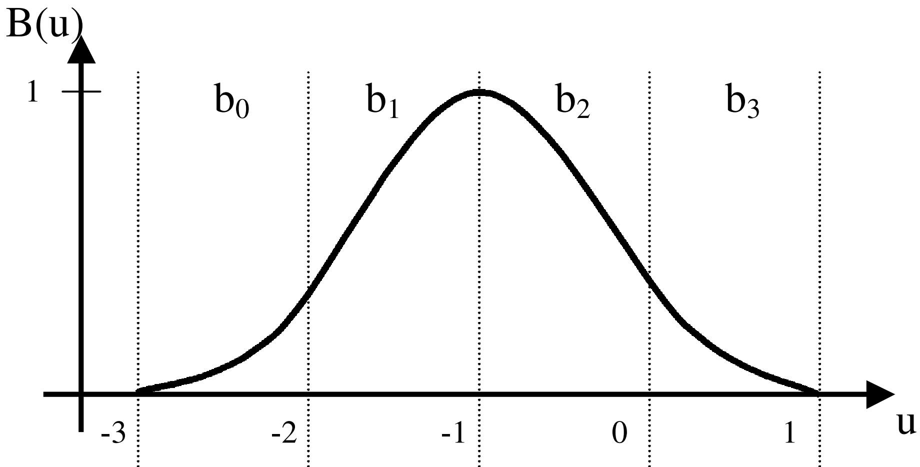
$$f(u) = \sum_{i=0}^{n-1} B(u-i)\, p_i \qquad (0 \le u \le n-3)$$

$$B(u) = \begin{cases} 0 & u < -3 \\ b_0(u+3) & -3 \le u < -2 \\ b_1(u+2) & -2 \le u < -1 \\ b_2(u+1) & -1 \le u < 0 \\ b_3(u) & 0 \le u < 1 \\ 0 & u \ge 1 \end{cases}$$

$$b_j(u) = \begin{cases} u^3/6 & j = 0, 0 \le u \le 1 \\ (1 + 3u + 3u^2 - 3u^3)/6 & j = 1, 0 \le u \le 1 \\ (4 - 6u^2 + 3u^3)/6 & j = 2, 0 \le u \le 1 \\ (1-u)^3/6 & j = 3, 0 \le u \le 1 \end{cases}$$

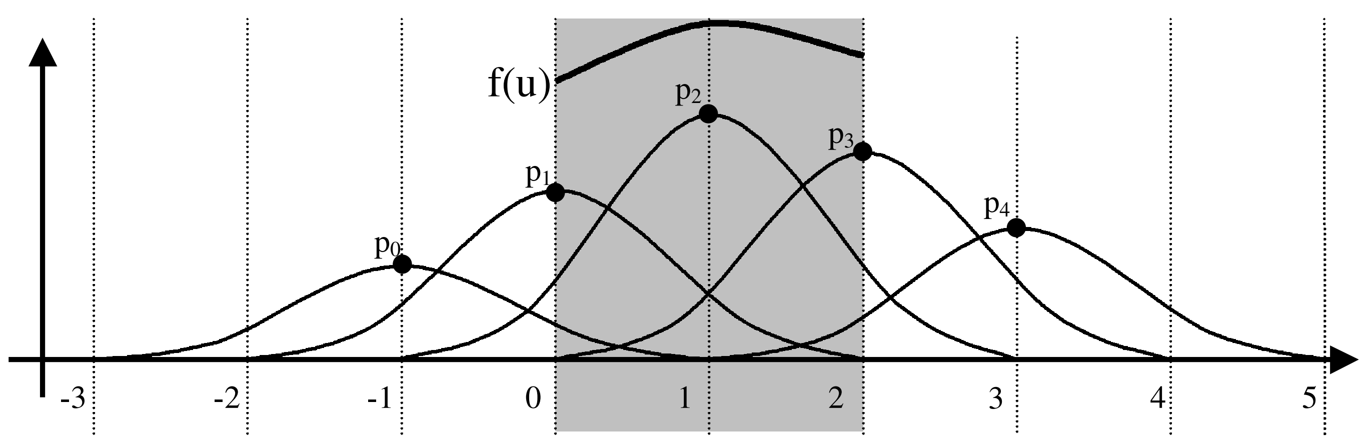
$p_0 \ldots p_{n-1}$ are the n control points. B(u) is the basis function that weights each control point.

Uniform Cubic B-Splines (cont'd):



- Each control point contributes to the spline, f(u), in 4 consecutive regions using one of the 4 blending functions, $b_j(u)$.
- A spline, f(u), is described by at least 4 control points, and defined where 4 basic functions overlap.

Example – A spline with 5 control points. Note that it is defined only in the shaded region where exactly 4 basis functions overlap.



How will you use all this?

- Allow the user to select the desired number of control points.
- Compute the control points that yield a curve that satisfies a least squares fit to the keyframe data for each DOF.
- Evaluate the splines by converting time to u (while determining the DOFs during an Idle() call)

Fitting the spline:

- Casting the concept of a spline fit into a matrix equation will reveal how to attack the problem.
- If the spline is f(u), $b(t_i)$ are the m keyframe values of a DOF at the times t_0 to t_{m-1}, p_j are the n control points, and u(t) is a function that converts time into u. Then ideally we desire that the following n equations hold:

$$f(u(t_i)) = b(t_i)$$

$$\sum_{j=0}^{n-1} B(u(t_i) - j)\, p_j = b(t_i)$$

Fitting the spline (cont'd):

$$\sum_{j=0}^{n-1} B(u(t_i) - j) p_j = b(t_i)$$

- We can write these m equations as a matrix equation by expanding the sum of n basis fn's:

$$\begin{bmatrix} B(u(t_0)) & B(u(t_0)-1) & \cdot & \cdot & \cdot & B(u(t_0)-(n-1)) \\ B(u(t_1)) & B(u(t_1)-1) & \cdot & \cdot & \cdot & B(u(t_1)-(n-1)) \\ \cdot & \cdot & \cdot & & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & & \cdot & \cdot \\ B(u(t_{m-1})) & B(u(t_{m-1})-1) & \cdot & \cdot & \cdot & B(u(t_{m-1})-(n-1)) \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ \cdot \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} b(t_0) \\ b(t_1) \\ \cdot \\ \cdot \\ \cdot \\ b(t_{m-1}) \end{bmatrix}$$

- This is a set of m equations in n unknowns.
- We call the matrix of time evaluated basis function values A, the vector of control points x, and the vector of keyframe values b.
- If m=n (i.e. same number of control points as keyframe data points), then we can find a single, unique solution for x by simply inverting A.
- Generally m>>n as our data sets can have hundreds of key frame data points, while a spline of some tens of control points will fit adequately.
- If m!=n, then these equations cannot be solved exactly, simultaneously.

Fitting the spline (cont'd):

- We seek, then, to find the solution, x, that minimizes the least squares error of the fitting spline to the given keyframe data.
- This amounts to finding a solution vector, x, that minimizes the following scalar quantity:

$$r \equiv \left| A \cdot x - b \right|$$

- This is precisely the quantity minimized by the solution, x, provided by a procedure called Singular Value Decomposition, SVD.
- We provide a SVD library that solves for the x vector given A and b.

Use SplineFit() and supply it with A and b to compute the control points. A[i][j] should be set to the weight (basis function value) of the j^{th} control point at the time of b[i].

Notice that in each row of A, only four consecutive elements are ever non-zero. This is a reflection of the property that each of the n-3 regions of f(u) is influenced only by 4 control points.

Animation – Idle()

- Figure out the wall clock time
- Recompute the values of the DOFs using an interpolation scheme.
- Update the matrices of the TreeNodes that have DOFs. There are six possible matrix types (TransformX/Y/Z and Rotate X/Y/Z).
- Each TreeNode has only one DOF
- glutPostRedisplay()

GetTime(int):

- TIME_INIT – Initialize the timer
- TIME_SINCE_FIRST – Return the number of microseconds since TIME_INIT
- TIME_SINZE_LAST – Return the number of microseconds since last call of any type.

Matrix Details:

- TreeNode matrices, node->xform, are type Mat.
- xform[i][j] refers to the i^{th} row and j^{th} column.
- Matrices are multiplied against ***column vectors.***

Optional Features:

Movie Files (.mpg, .avi, etc.):

- Animate using constant time steps
- At each time step, use glReadPixels() to copy the frame buffer and save it to a file
- Convert the resulting sequence of screen shots into a movie using *movieconvert* on the SGIs

Camera Path Splines:

- Save a set of 3D control points into some file format of your creation
- Move and orient the camera along this path in real-time and/or during movie creation.
- For example: camera position follows the 3-D spline value, and the orientation follows the Frenet frame.

Re-sampling:

- Fit a spline to the keyframe data
- Replace the keyframe data with by evaluating the spline (perhaps at different time resolution)

Transitioning:

- Just evaluate the spline for the first .key file until #START_TIME of the second file
- Between #START_TIME of second file, but before #DURATION of first file elapses, linearly blend the evaluations of both splines
- After #DURATION just evaluate the spline for the second .key file.

Compositing:

- Just evaluate the spline for the base .key file until #START_TIME of the subset file
- Between #START_TIME and #DURATION of the subset file, linearly blend in and then out the motion of the subset of joints in the subset .key file.

Keyframe File Creation:

- Let the users step through the key frames, select DOFs, and adjust them.
- Implement key frame copying to assist you in creating animations

Looping:

- Fit a *closed* spline rather than an open spline
- For open splines we had the n control points: $p_0, p_1, p_2, \ldots, p_{n-1}$
- To close a spline extend it by three control points that are copies of p_0, p_1, p_2
- Now we have n+3 control points as follows: $p_0, p_1, p_2, \ldots, p_{n-1}, p_0, p_1, p_2$
- To fit such a spline keep in mind:
  - There are now n+3 control points, so u runs from $u = 0$ to $u = (n+3) - 3 = n$ [not n-3]
  - There are still only n *unique* control points, and thus only n columns in matrix A and n rows in vector b
  - Map the 'virtual' columns n, n+1, n+2 of A back onto columns 0, 1, 2 to reflect that the three additional control points are copies of the first three control points!
- After fitting, you can go ahead and make genuine copies of p_0, p_1, p_2 into p_n, p_{n+1}, p_{n+2}
- You might apply this to the walk cycle (but throw out the translation DOFs)