P. Maes. The mit alive project. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 1993.

Claudia L. Morawetz and Thomas W. Calvert. Goal-directed human animation of multiple movements. *Proc. Graphics Interface*, pages 60–67, 1990.

K. Perlin. An image synthesizer. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):287–293, 1985.

K. Perlin. Danse interactif. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 28(3), 1994.

M. Raibert and et al. *Legged Robots That Balance*. MIT press, 1986.

SGI. *SGI Programmers Manual*. Silicon Graphics Incorporated, Mountainview, 1994.

Karl Sims. Evolving virtual creatures. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 28(3):15–22, 1994.

Neal Stephenson. *Snow Crash*. Bantam Doubleday, New York, 1992.

the impression of social complexity. For example, could one structure entire narratives in this manner?

5 Conclusions

Using ideas from procedural texture synthesis, we are able to create remarkably lifelike, responsively animated characters in real time. By conveying just the "texture" of motion, we are able to avoid computation intensive dynamics and constraint solvers. We believe these techniques have the potential to have a large impact on computer Role Playing Games, simulated conferences, "clip animation", graphical front ends for MUDs, and synthetic performances.

6 Acknowledgements

I would like to thank Athomas Goldberg for production support on this paper, and in particular for the illustrations, as well as on the work itself. I would also like to thank Cynthia Allen, David Bacon, Troy Downing, Mehmet Karaul, Tom Laskawy, Kuochen Lin, Jon Meyer, and Jack Schwartz for all their help and encouragement. Ben Bederson, Bruce Naylor, and Silicon Graphics Inc. have provided hardware assistance for this research. Thanks as well to Marcelo Zuffo, Roseli Lopez, and the folks down at the University of Sao Paulo for all their support. And *muito obrigado* to Emi, who inspires the dance.

7 References

Norman I. Badler, Brian A. Barsky, and David Zeltzer. *Making Them Move: Mechanics, Control, and Animation of Articulated Figures.* Morgan Kaufmann Publishers, San Mateo, CA, 1991.

N.I. Badler, C. Phillips, and B.L. Webber. *Simulating Humans: Computer Graphics, Animation, and Control.* Oxford University Press, 1993.

R. Brooks. A robust layered control system for a mobile robot,. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

J.E. Chadwick, D.R. Haumann, and R.E. Parent. Layered construction for deformable animated characters,. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):243–252, 1989.

D. Ebert and et. al. *Texturing and Modeling, A Procedural Approach.* Academic Press, London, 1994.

D. Gerlernter. *Mirror Worlds.* Oxford University Press, 1992.

M. Girard and A.A. Maciejewski. Computational modeling for the computer animation of legged figures. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 20(3):263–270, 1985.

A related notion that we will explore is peripheral attention. For example, suppose a man and a woman are engaged in conversation, and another man appears in the line of vision of the first. How would one show the fact that the effect of the woman's attention involuntarily drifting toward the other man, even though her intention is to maintain the conversation? Similarly, how would one show the shift in each man's attitude when the woman's behavior is noticed? The first man might begin to talk more frequently; the second might drift toward the conversation.

In recent work, we have developed methods of running these group simulations on multiprocessors and across multiple networked computers. This is done over a network of UNIX workstations as follows. Each actor is a separate program which communicates through its standard input and standard output. A supervisory rendering process opens up a two way read/write pipe to each actor. Each actor may be invoked via a remote shell, so that it need not be on the same workstation as the renderer. To send messages, actors print commands to their standard output which are parsed and executed at each frame by the supervisor program. If one actor wants to send a message to another, then it prints a wrapper command. This wrapper command instructs the supervisor program to print a message command string to the standard input of the recipient actor. The recipient then parses and executes this message.

This approach makes it quite easy to allow different kinds of actors to each respond in the most appropriate way to a given message. The Camera is an actor which possesses behavior like any other. For example, in our current system Actor1 can send the messages:

```
{ my_location "look_here" } "Camera" send_message
{ my_location "look_here" } "Actor2" send_message
```

where *my_location* is the current x, y, z position of Actor1. Note that different recipients are free to interpret any message as they see fit. For example, the Camera actor generally averages all "look_here" requests, so that it keeps all attention seeking actors in its range of vision. In contrast, if several "look_here" requests are sent to a human actor, it will generally honor only one of them. As a result, an actor will adjust his/her gaze to track only the message sender of greatest interest. This provides a simple object-oriented message capability, with overloading of methods based on the type of the recipient.

In addition, we are exploring immersive interactions using projector screens and position sensors, so that real people can interact with these characters, which are digitally composited into miniature models of interior spaces. Within this experimental laboratory we explore questions of how to convey peripheral awareness, approach/avoidance, "paying attention", "listening", etc. This is in the spirit of the recent *Alive* project of Maes' et al at MIT (Mae93). We are particularly interested in immersive scenarios involving two or more projection screens, in order to see to what extent simulated body language will help to convey the impression of various competing social or attention getting activities.

We are also studying the semantics of the discrete state transitions that visually represent shifts in attitude and attention. We are particularly interested in determining to what extent can we encode merely the rhythm of interpersonal interaction, in order to convey

2.10 User interaction

User interaction is quite simple. The user only needs to control the discrete states of the puppet. Currently this is handled by a panel of buttons [figure 12]. All continuous behavior is automatically derived by the system through integration over time, as previously described.

3 Implementation

In our current instantiation, all parts are rendered as polygonal mesh approximations of 50 ellipsoids. The simulation has been run on an SGI Indigo Elan (at 7.5 frames/sec) or an Indigo 2 (at 15 frames/sec), and calls the SGI GL library for rendering.

It also runs efficiently on any UNIX or 486 based LINUX machine, but for this instantiation the figure can be rendered only in silhouette. In this case rendering is done by computing the silhouette ellipse for each ellipsoid and then doing a software scan conversion [figure 13]. Since this is a silhouette rendering, front-to-back ordering does not need to be taken into account. Using this method, the dancer runs at 6 frames per second on a 486/DX66 processor.

The button panel is implemented via a small stand-alone tcl/tk program. Communication with the this program is done through a two-way ascii pipe.

4 Ongoing and Future work

We are looking at the combination of these procedural techniques with motion capture. The research question here is how to analyze motion capture of walk cycles or gestures in order to convert them into a form compatible with the procedural synthesis techniques. Our approach is to align the natural cycles of walks and other rhythmic motions so that they can be blended together.

We also are beginning to study group interactions between these simulated puppets. This research is focused on situations in which people communicate richly through body language, such as parties, bar scenes, meetings.

Because all control of a puppet's state is discrete, knowledge of actions and transitions between two or more interacting puppets can be done by exchanging state tokens. If a character knows the state of another character, then for non-contact pairwise interactions it suffices to know only the position and facing direction of the other character. This method of communication is fast and compact, and scales up gracefully in simulations with large numbers of puppets.

Using this approach, we make the characters "press each others' buttons." For example, when two characters are engaged in conversation, they tend not to simultaneously talk at once (although they occasionally do), and they also tend to avoid long collective silences. When a third character walks up the behavior of the other two shifts, depending upon the status of the newcomer and how each of the other two feels about him/her.

Object avoidance is done as follows. Each wall emits a repulsive force vector, which increases near the wall. We sum all of these vectors. If the puppet walks into such a vector field, and is angled off to the left (or right) of facing the wall, then we give her a tendency to turn more to the left (or right). When this is tuned properly, she just avoids walls, and so we don't have to worry about collisions. In the more general case, we would put a similar repulsive vector field around any object we want her to avoid, as well as an attractor field at each open doorway. This would act as a variety of remote compliance to help her find her way in.

We can make the puppet "look at the camera", just by turning her Neck joint. Actually she can look at any aim point in the scene. But this is not desirable when the puppet's body is facing directly opposite from this direction. To avoid complete backward head turns, we add a constraint into the neck turning joint. A dot product of the body's forward position and the desired aim direction is calculated. As the value of this dot product drops from 1.0 down to -1.0, we continually lessen the factor by which we influence the head to turn in the aim direction. We found through trial and error that we get the most natural results when this factor reaches zero at a dot product value of -0.6.

The visual effect is that as the puppet turns away from us, she holds our gaze for a bit, and then gradually ignores us as she continues to turn further away. Then as she continues turning, she eventually locks her gaze with us again on the other side, by turning her head over her other shoulder [figure 10].

The above constraints constitute all of the "physics" built into the model, other than the natural constraints imposed by the forward kinematics itself (ie: that the limbs never fly apart).

2.9 Shifting body parts

In order to achieve real-time performance, it is important to limit the elaboration of puppet geometry (see next section). Yet we do not wish to sacrifice the appearance of human form. To attain a natural appearance without using large numbers of body parts, we shift body parts around as joints flex, in order to keep the visual appearance of human form for all body angles.

For example, the thigh consists of three intersecting ellipsoids, one for the main thigh mass, a second for the muscle in back of the thigh, and a third for the muscle high up and inside the thigh. When the puppet bends the thigh very far backward about the hip (such as in a fondue turn [figure 6]) the two front ellipsoids have a tendency to separate from the puppet's pelvis. We compensate as follows. As the hip joint bends backwards, we slide these ellipsoids *down* the thigh (away from the hip) in linear proportion to the degree of bending [figure 11]. We provide similar sliding mechanisms at all body parts where such compensation is needed.

zero up to one or back down again. This is specified in seconds, not frames - behavior should not change with frame rate! In order to effect this, we use the actual system clock, not a frame counter, to time transitions. We also use the system clock to drive the signal sources described above in section 2.3.

Some transitions look better when they are fast, and others look better when slow. It is surprising how much expressiveness one can achieve by tuning these transition times. For example, when the dancer puppet performs the action "put hands on hips indignantly and look at the camera", she puts her hands on her hips first, and only then, a beat later, does she turn to look at the viewer [figure 9]. Then when she goes from this state into the slow dance, she continues to look at the viewer for a second or so, even while she's already dancing and her body is turning away.

We conjecture that this behavior looks correct because fixing one's gaze on another person is a more explicit emotional signifier than is changing one's bodily activity, and therefore should happen more slowly. In this case, even when she is beginning to dance we still want the dancer to convey the reminder that she was annoyed at us just a moment ago.

We believe that the use of different transition times for various parts of a gesture is a very powerful means for conveying subtle impressions of intention. Although our approach to this is currently ad hoc, we hope that experimentation of this kind can lead to a set of useful rules for understanding of human body language. Related work in the role of emotion and communication in gesture is found in the chapter by Calvin and Morevic in (BBZ91).

2.8 Non-hierarchical motions

The puppet will always conform to certain simple constraints no matter what the forward kinematics specify. Here are the key constraints:

- the foot on the ground propels the puppet
- the supporting foot must be at floor level
- obstacles are avoided by turning away from them
- the head won't turn all the way around backwards

Each of these constraints is imposed by a few lines within the code that models the body. For example, to propel the puppet from the foot we detect the lowest foot. We measure how far this foot has moved since the previous frame. We then add this to a cumulative displacement, and apply a positional offset to the rendered body equal to the opposite of this displacement. The effect is that the lower foot always stays in place and propels the body.

Also, whenever we compute each new total foot position, we average in half of the previous total just for the y (vertical) component. The effect of this is to always keep the supporting foot level with the ground.

2.6 Dependencies between weights

Let's say that the puppet is walking, and we decide to have her do a pirouette. It would make no sense for her to continue walking while she is pirouetting. The mechanism we employ is to build dependencies between weights.

The pirouette weight acts as an inhibitor - as its value rises from zero to one, it drives down the effect of the weight that controls such steady state actions as walking. Then as the pirouette weight drops down to zero again, the walking weight is allowed to take effect again.

The numerical value of the walking weight is not itself modified. But anything that depends upon it is seen through the filter of the pirouette weight. Conceptually, the pirouette weight "blocks" the walking weight, much as the alpha channel of a foreground image blocks a background image during a compositing operation [figure 7].

Note that an action need not involve all joints. Examples include such actions as waving with the left arm, shrugging the shoulders, or scratching one's head. If an action which involves only a subset of the joints blocks another action, then it will only block those joints included in this subset. So we may use this structure to "layer" partial actions. For example, a puppet that is running can be told to wave his hand, without breaking his stride.

A small section in the program creates a layering structure for such dependencies. This control is divided into two levels - *states* and *weights*. The state level consists entirely of discrete boolean values. For example, either the puppet "wants" to walk, or she does not. The weight level consists of continuous values between zero and one. These are derived by integrating the effect over time of the discrete states [figure 8]. These continuous weights are what go into the convex sum above, to drive the puppet. Some of these weights are dependent on others.

For example, if a user directs the puppet to walk, then the discrete walk state turns on, and the continuous weight controlling the walking action gradually rises from zero to one. If the user subsequently directs the puppet to perform a pirouette, the weight of the pirouette action gradually rises to one. The discrete walk state continues to stay on, but the continuous weight of the walk is driven down to zero by its dependency on the weight of the pirouette action. When the pirouette state is disabled, the weight of the pirouette action gradually falls to zero, and the walking action at the joints gradually reappears.

The dependencies between weights are implemented by a sequence of conditional expressions. This approach is similar in spirit to Brooks' subsumption architecture (Bro86) for walking robots, in which more immediate goals (eg: "don't fall over") block out longer term goals (eg: "walk to the edge of the table").

2.7 Transition times

Each user specified action starts the rise of some scalar weight from zero to one, via an S-shaped ramp. We have found that the only tuning needed for controlling the shape of any given transition is a single scalar value that specifies the duration of the transition from

2.4 Combining multiple weighted actions

To combine actions we assign numerical weights to every potential action in the action mix. These weights give the relative contribution of each action to the total motion of the puppet. The weights vary over time; at any given moment, only a few actions have a non-zero weight. For each joint, the contributions from all the actions to that joint's position are combined via a convex sum (a weighted sum in which the weights add to unity). We do this as follows.

Assume there are k actions with associated weights $w_1, w_2, ...w_k$, and that there are n universal joints in the entire body, where each joint involves three rotational degrees of freedom $[x, y, z]$. Any one of the k actions will use only some subset of these n joints. Let C_i be a vector whose values are 1 for each joint j used by action i, and 0 otherwise.

Let $A_i = [V_1, V_2, ...V_n]$ be a vector representing the value $[x, y, z]$ that is generated by action i for every universal joint j. We obtain the position of universal joint j via: $\sum (A_{ij} C_{ij} w_{ij}) / \sum (C_{ij} w_{ij})$.

In the next sections we describe the way in which we coordinate the variation over time of the weights of different actions.

2.5 Synchronizing actions

The phases of all the actions are synchronized. For example, when a dancer puppet is walking, and we ask her to do a classical fondue turn to the right, she won't begin the turn until she is about to put her weight down on her right foot [figure 6]. This is the only sensible thing to do, since one must begin a fondue turn by stepping into it. This requires that both the walk and the fondue turn are built from expressions that run off the same master clock.

We handle transitions between two actions that we wish to have different tempos via a morphing approach: At the start of the transition, we use the tempo of the first action; at the end, we use the tempo of the second action. During the time of the transition, we continuously vary the speed of the master clock from the first to the second tempo. In this way, the phases of the two actions are always aligned during transitions.

We may also define new actions as extended transitions between two or more other actions. For example, we may morph between the "running" example above and a "standing at attention" pose, in which all joint angles are fixed at zero. When the interpolant is in the range 0.3 to 0.5, this interpolated action becomes a visually realistic walk (to the author's surprise). But a human walking tempo is also 0.3 to 0.5 that of a human running tempo. For this reason, we use the morph transition parameter to modulate the tempo. In this way a puppet can be made to continually and realistically transition from standing still, through walking, to running, or to anywhere in between.

In a scene with multiple puppets (see section 4 below) each puppet maintains its own individual tempo.

```
    c1      => A
    s1      => B
    1 A -   => C
    1 B -   => D
    c2      => A2
    1 A2 -  => B2

    {    0 -15   0 } {   5 15 0 } { A2          C 0  } Waist
    {    0 -90   0 } {   0 90 0 } { 0           N 0  } Head
    {    0 -10  -5 } {   0 10 5 } { 0           C D  } Rchest
    {    0   0   0 } {  45  0 0 } { D           0 0  } Rshoulder
    { -120   0 -10 } {   0  0 0 } { C           0 B2 } Relbow
    {  -10   0   0 } {  10  0 0 } { C           0 0  } Rwrist
    {    0 -10  -5 } {   0 10 5 } { 0           A B  } Lchest
    {    0   0   0 } {  45  0 0 } { B           0 0  } Lshoulder
    { -120   0 -10 } {   0  0 0 } { A           0 B2 } Lelbow
    {  -10   0   0 } {  10  0 0 } { A           0 0  } Lwrist
    {    0 -10   0 } {   0 10 0 } { 0           A 0  } Rpelvis
    {  -40   0   0 } {  40  0 0 } { A B .3 * -  0 0  } Rhip
    {    0   0   0 } { 130  0 0 } { B .2 bias   0 0  } Rknee
    {  -45   0   0 } {  45  0 0 } { C .7 bias   0 0  } Rankle
    {    0 -10   0 } {   0 10 0 } { 0           C 0  } Lpelvis
    {  -40   0   0 } {  40  0 0 } { C D .3 * -  0 0  } Lhip
    {    0   0   0 } { 130  0 0 } { D .2 bias   0 0  } Lknee
    {  -45   0   0 } {  45  0 0 } { A .7 bias   0 0  } Lankle
} 'running define_action
```

This action consists entirely of rhythmic motion, except for the head's "looking around" movements, which are driven by coherent noise. The double speed oscillations serve to slightly bend and unbend the waist twice per cycle, as weight is alternately borne either by one foot or by two feet. Double speed oscillations are also used to rotate slightly about the elbow's y axis. This rotation pulls the forearms a bit closer to the body twice per cycle - both when in front and when behind the body.

Note the use of the bias function in the rotations about the knee joints. These give the non-weight bearing knee a little extra kick at the time it swings forward. For the same reason, we add a small amount of rotation to each hip, 90 degrees out of phase with its primary motion.

Once an action is designed, this sort of structure provides many opportunities for customization. In the above example, we can replace each of the constants in the knee and hip "bias" expressions by variables. As we modify these variables we obtain walks that reflect different emotive states and degrees of energy.

the limits of rotation about the x axis are -15 degrees to 15 degrees. Similarly, the y axis is fixed at 0 degrees, and the z axis varies from 5 degrees to -5 degrees.

The time varying behavior for this joint is as follows. The x axis interpolates between its limits as $c1 = rcos(time)$ and the z axis interpolates between its limits as $s1 = rsin(time)$. The y axis stays fixed.

The action defined above is a relatively stylized dance step, so most of its motion is rhythmic, controlled by periodic functions. Only the head motion has a little randomness, which in this case gives the impression that the puppet is looking around while she dances.

Notice also that the joints at the chest are driven by $c1$ in their x axis and by $s1$ in their z axis. This gives an elliptical motion to the shoulders, which is crucial for giving the subtly "latin" feel of this dance move.

In contrast, here is the definition for standing in a casual pose [figure 5]:

```
{
   {   0 15   0 } { 0 -15 0 } {  0 n1 0 } Neck
   {  20  0   0 } {         } {         } Nod
   {   0  0  -5 } {         } {         } Lchest
   {   0  0   0 } {         } {         } Rchest
   { -10  0   0 } {         } {         } Lshoulder
   { -10  0   0 } {         } {         } Rshoulder
   {   0  0 -10 } {         } {         } Lelbow
   {   0  0 -10 } { 0  0 -5 } {  0 0 n1 } Relbow
   {   0  0   5 } {         } {         } Waist
   {  -2  0   2 } { 2  0 -2 } { n1 0 n1 } Lpelvis
   {  -2  0  -2 } { 2  0  2 } { n1 0 n1 } Rpelvis
   {   0  0 -14 } {         } {         } Lhip
   { -10 25  12 } {         } {         } Rhip
   {  -5  0   0 } {         } {         } Lknee
   {  25  0   0 } {         } {         } Rknee
} 'stand define_action
```

Here most of the vectors are left blank. This means that these joints are completely static for this action. All of the motion of this action is driven by noise - there is no rhythmic motion at all. The noise gives the effect of subtle restlessness and weight shifting. The motion is subtle, but if it is left out, the puppet looks stiff and unrealistic.

For some actions, we put small additional expressions in the third, time dependent, vector in order to couple actions between the joints, or to modify the bias or gain of a joint (Per85). For example, here is the specification of a running action. For clarity of exposition, we have assigned the four phase signals to variables A, B, C and D, respectively. We have also assigned double speed oscillation and its complement to variables $A2$ and $B2$, respectively.

```
{
```

(1) if x is an integer, $noise(x) = 0$.

(2) define a mapping $G(i)$ from the integers to a fixed set of pseudorandom gradients.

(3) given any $i < x < i + 1$, do a hermite spline interpolation, using the two neighboring gradients $G(i)$ and $G(i + 1)$.

The only tricky step above is (2). To implement this step efficiently, we precompute a table of pseudorandom gradients $g[0..255]$. Then for any integer i we return $g[i_{mod256}]$. A comprehensive discussion of noise implementations can be found in (Eea94).

Every action is built by using some combination of the above source signals in simple expressions, to control the rotation of some joints within some range. Each joint is at its zero position when the puppet is standing at attention with both arms at the side. An action is specified by a table of ranges and time dependent behavior for each joint that this action affects.

The stylized way in which these are coded makes it simpler to provide high level descriptions of rhythmic motions. Here is the code we use to specify a "rhumba" dance [figure 4]:

```
{
   {    5   5   5   } {   -5  -5  -5 } { n1 n2 n3 } Nod
   {   15   0   5   } {  -15   0  -5 } { c1  0 s1 } Rchest
   {    0   0   0   } {    0   0   0 } { s1  0 s1 } Rshoulder
   {  -90   0   0   } {  -70   0   0 } { s1  0 s1 } Relbow
   {    0   0   0   } {    0   0   0 } { s1  0 s1 } Rpelvis
   {  -25 -15   5   } {    0   0 -10 } { s1 s1 s1 } Rhip
   {   50   0   0   } {    0   0   0 } { s1  0 s1 } Rknee
   {    0   0   0   } {    0   0   0 } { s1  0 s1 } Rankle
   {    0   0  10   } {    0   0 -10 } { s1  0 s1 } Waist
   {  -15   0  -5   } {   15   0   5 } { c1  0 s1 } Lchest
   {    0   0   0   } {    0   0   0 } { s1  0 s1 } Lshoulder
   {  -70   0   0   } {  -90   0   0 } { s1  0 s1 } Lelbow
   {    0   0   0   } {    0   0   0 } { s1  0 s1 } Lpelvis
   {    0   0 -20   } {  -10 -25  20 } { s1 s1 s1 } Lhip
   {    0   0   0   } {   20   0   0 } { s1  0 s1 } Lknee
   {    0   0   0   } {    0   0   0 } { s1  0 s1 } Lankle
} 'rhumba define_action
```

Each line of the above code specifies an assignment of three items to a particular joint (Nod, Rchest, etc). Each of these items contains three numeric values. Each of the first two items is run immediately and packaged up as a vector, representing an extreme position of motion for the joint. The third item is evaluated at every frame where the action is performed, and is used as a linear interpolant between the two extremes.

Let us take the second line as an example. It specifies motion for the "Rchest" joint, the universal joint which pivots around the chest to displace the right shoulder. For this joint,

```
      Rhip    joint
      Rknee   joint
      Rankle  joint  2 draw_leg
   Pop
Pop
```

Push and Pop manipulate a local matrix stack, as in the Silicon Graphics GL model (SGI94). Waist, Neck, Nod, Lchest, etc., are joint variables, and draw_head, draw_arm, draw_torso, draw_leg are procedures to draw parts of the body. Note the scaling by -1 in the x dimension to draw the right arm and leg as mirrors of the left arm and leg.

The variables Waist, Neck, Nod, etc., represent the 19 universal joints of the figure. Each is a vector of length three. The values in these vectors, which change at every frame, drive the figure's joints. They are set by actions and by transitions between actions.

The actual work is done in subprocedures draw_head, draw_torso, draw_arm and draw_leg. Each of these routines does successive forward kinematic transformations on the current matrix in order to compute locations for the puppet's component parts.

2.3 Actions

A primitive action is constructed by varying the puppet's scalar joint angles over time t via expressions of raised sine and cosine, as well as noise, where "raised sine" and "raised cosine" are defined by $\frac{1+sin\theta}{2}$ and $\frac{1+cos\theta}{2}$.

At each frame we compute raised sine and cosine, as a function of time, at two frequencies one octave apart:

```
s1 := rsin(time)
c1 := rcos(time)
s2 := rsin(2*time)
c2 := rcos(2*time)
```

The variables $s1$ and $c1$ are used together within actions to impart elliptical rotations. Variables $s2$ and $c2$ do the same at double frequency.

Together the expressions $s1$, $c1$, $(1 - s1)$, and $(1 - c1)$ collectively generate a four phase periodic signal. In practice we have not found any need for finer phase control of periodic actions than this quarter cycle accuracy.

We also provide a set of independent coherent noise sources $n1, n2, ...$:

```
n1 := .5 * (1 + noise(t))
n2 := .5 * (1 + noise(t + 100))
n3 := .5 * (1 + noise(t + 200))
```

The coherent noise source is defined as in (Per85). Here the noise is simpler, since it need only be defined over a one dimensional temporal domain, rather than over a three dimensional spatial domain. The algorithm we use is:

layers of choice controls

constraints

2.2 Bottom level kinematic hierarchy

19 universal joints are used in our current approximation of the human figure: one each for waist, neck, and head, plus four for each limb [figure 3]. Ideally there should be more; this was the minimum that seemed necessary to allow emotional expressiveness.

Separate joints appear at the base and top of the neck. The first universal "arm" joint is actually at the chest. This controls the position of the shoulder. The arm structure involves the following universal joints: chest, shoulder, elbow, and wrist. Similarly, a leg has: pelvis, hip, knee, and ankle joints.

Each universal joint allows three rotations: x, then z (the two "aiming" parameters), followed by y (rotation around the limb axis). Any angle not specified defaults to a "zero" position where the puppet is standing upright with arms at her side.

Here is the actual code of the main routine for positioning and drawing the body, executed once per frame, as expressed in our modeling language's reverse polish notation:

```
Push
   Neck joint
   Nod  joint draw_head

   Push
      Lchest    joint
      Lshoulder joint
      Lelbow    joint
      Lwrist    joint draw_arm

      -1,1,1 scale
      Rchest    joint
      Rshoulder joint
      Relbow    joint
      Rwrist    joint draw_arm
   Pop
   Waist draw_torso
   Push
      Lpelvis joint
      Lhip    joint
      Lknee   joint
      Lankle  joint  1 draw_leg

      -1,1,1 scale
      Rpelvis joint
```

movement of tripping and recovering its balance, unless we've already taught it how to do that.

The two approaches are compatible. Ideally a system would employ both predetermined movements, as well as physical laws that allow it to deal with unexpected events in its environment, such as the sorts of dynamic balancing of the 'Jack' figure from Badler's group at U. Penn.

2 Method

2.1 Actions, Weights, Transitions

The two major components of our method are *actions* and *weights*. An *action* is some simple or repetitive movement, such as walking or standing, or a pirouette turn. The relative contribution of an action is given by a *weight*, which is always a scalar value between 0.0 and 1.0.

We cause the puppet to respond to her environment in real time mainly by changing these various weights. For example, decreasing the weight for one action, while simultaneously increasing the weight for another, causes a transition in behavior from the first action to the second [figure 1].

Note that if this is done properly, the actions can be connected together seamlessly in arbitrary sequences, like characters in a string of text, to build up complex behaviors [figure 2]. In spirit this is similar to the summation of B-spline knot functions to construct smooth piecewise-cubic curves.

If these transitions are applied naively, the results can be disastrous. An example would be a transition from an action in which our puppet is stepping onto her left foot to an action in which she is stepping onto her right foot. Another example would be a transition that would make the puppet's arm interpenetrate her body on its way from the first action to the second action.

We solve this problem by controlling the times of transitions between actions, and by designing actions in such a way that it is possible to control their transitions.

In this section we will describe the structure of our system, proceeding in a bottom up fashion. We start with joint kinematics, go on to explain how coherent actions are developed, and then to the scalar controls that combine those actions. The bottom to top structure of our system is as follows:

joint kinematics

individual actions

scalar weights to blend actions

synchronizing actions

discrete choice controls

In the recent work described in this paper we have applied this approach to the problem of building real time graphic puppets that appear to be emotionally responsive.

We choose the word "puppets" very deliberately here. This work is not artificial intelligence - these animated characters do not encode any real intentionality; they only encode a visual impression of personality. This work was first presented in (Per94). We will refer to the "dancer" figure from that animation in our examples.

1.2 Related work

Simulated actors that embody true physical constraints are being developed by Badler et al at the University of Pennsylvania (BPW93). Dynamic balancing walking robots have been developed by Raibert (Rea86), and animal and human figure simulations based on inverse dynamics have been developed by Girard (GM85).

Using layered construction in the design of articulated movement has been explored by Chadwick et al (CHP89). Similarly, layered control structures for walking robots (subsumption architectures) have been used effectively by Brooks (Bro86).

Morawetz and Calvert have added a sense of personality to simulated human movements by supporting secondary movements (MC90). In a radically different approach, genetic algorithms have been seen to induce movement that gives an intriguing impression of personality in goal directed mutations of articulated figures (Sim94).

1.3 Guiding principles

We adopt the following general approach: Program individual motions into the puppets beforehand, but also ensure that transitions between any pair of actions are visually correct.

A potential objection to this approach is that things might look repetitious. But by using randomization we can easily build actions that are very controllable yet never actually repeat themselves.

In addition to the forward kinematics of individual actions, we build in only three simple constraints. Characters never walk through walls, they never spin their heads all the way around backwards, and they maintain fixed foot contact with the floor when doing "walking" actions.

1.4 Comparison with dynamics approaches

This approach has advantages as well as disadvantages when compared with more ambitious approaches that try to model the underlying physics. One advantage is that it allows much more direct control over the subtle movements that convey the appearance of emotional expressiveness. Another is that computational costs are far lower.

A disadvantage is that the model can't teach itself new actions. If the puppet's foot is snagged by a rock while walking, the puppet cannot properly perform the particular

Real time responsive animation with personality

Ken Perlin
Media Research Laboratory
Department of Computer Science
New York University
715 Broadway, NY, NY 10003
perlin@nyu.edu

Abstract

Building on principles from our prior work on procedural texture synthesis, we are able to create remarkably lifelike, responsively animated characters in real time. Rhythmic and stochastic noise functions are used to define time varying parameters that drive computer generated puppets. Because we are conveying just the "texture" of motion, we are able to avoid computation of dynamics and constraint solvers.

The subjective impression of dynamics and other subtle influences on motion can be conveyed with great visual realism by properly tuned expressions containing pseudo-random noise functions. For example, we can make a character appear to be dynamically balancing herself, to appear nervous, or to be gesturing in a particular way.

Each move has an internal rhythm, and transitions between moves are temporally constrained so that "impossible" transitions are precluded. For example, if while the character is walking we specify a dance turn, the character will always step into the turn onto the correct weight-bearing foot. An operator can make a character perform a properly connected sequence of actions, while conveying particular moods and attitudes, merely by pushing buttons at a high level.

Potential uses of such high level "textural" approaches to computer graphic simulation include Role Playing Games, simulated conferences, "clip animation", graphical front ends for MUDs (Ste92) (Ger92), and synthetic performances.

1 Introduction

1.1 Description of the Problem

In previous work (Per85) we used pseudo-random functions to create natural surface textures of surprisingly realistic appearance without having to model the underlying physics. This was done with a set of interactive tools, consisting of:

a powerful interactive prototyping language

a good set of signal generation and modification functions

a good controllable noise primitive

a set of conventions for fitting things together