The Design and Engineering of Variable Character Morphology

Scott Michael Eaton
B.S. Mechanical Engineering
Princeton University, May 1995

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 2001

Author

Scott Michael Eaton
Program in Media Arts and Sciences
August 10, 2001

Certified by

Bruce M. Blumberg
Associate Professor of Media Arts and Sciences
Asahi Broadcasting Corporation Career Development
Professor of Media Arts and Sciences

Accepted by

Dr. Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

The Design and Engineering of Variable Character Morphology

Scott Michael Eaton

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 2001

Read by:

[illegible]

Rebecca Allen
Professor
Department of Design | Media Arts
University of California Los Angles

[illegible]

Hisham Bizri
Research Fellow
Center for Advanced Visual Studies
Massachusetts Institute of Technology

The Design and Engineering of Variable Character Morphology

Scott Michael Eaton

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 2001

Abstract

This thesis explores the technical challenges and the creative possibilities afforded by a computational system that allows behavioral control over the appearance of a character's morphology. Working within the framework of the Synthetic Character's behavior architecture, a system has been implemented that allows a character's internal state to drive changes in its morphology. The system allows for real-time, multi-target blending between body geometries, skeletons, and animations. The results reflect qualitative changes in the character's appearance and state. Through the thesis character sketches are used to demonstrate the potential of this integrated approach to behavior and character morphology.

Thesis Supervisor: Bruce Blumberg
Title: Asashi Broadcast Corporation
Career Developmental Professor
of Media Arts and Sciences

acknowledgements

I would like to thank Prof. Bruce Blumberg for giving me the opportunity to study and work with the Synthetic Characters Research Group for the past two years, the experience has been challenging and rewarding.

Thanks to Robert Burke for the numerous collaborations over the past two years. Whether constructing animatronic dog heads, evolving ants, or making two-hundred foot mutants, it was all fun.

Thanks to Marc Downie, my one-time officemate and technical advisor, for endless advice and patience.

And thanks to the rest of the group with whom I have had the pleasure of working: Damian Isla, Ben Resner, Michael P. Johnson, Bill Tomlinson, Jed Wahl, Dan Stiehl, Geoff Beatty, Matt Berlin, Jesse Gray, Yuri Ivanov, and Adolf Wong.

Special thanks to my parents and AnnMarie Doherty for their love and support.

The Design and Engineering of Variable Character Morphology

Contents

List of Figures

Chapter 1
Introduction

Morphology *n.* I. The science of form. II. *Biol.* That branch of biology concerned with the form of animals and plants, and of the structures, homologies, and metamorphoses which govern or influence that form. [1]

As defined above, morphology is concerned with the study of form. Within a biological context, morphology and functional morphology investigate the structure, development, and purpose of form within living organisms. These concepts have immediate and pertinent analogs to artificial creatures. As computational behavior systems become more complex and develop persistence models that develop over long periods, artificial creatures will need to possess developed, morphologically complete bodies that allow them to grow, age, and change over time. This thesis investigates the engineering required to endow virtual creatures with bodies capable of developing in this way.

1.1 A General Morphology System

The core contribution of this thesis is a virtual *morphology system* that gives a character the ability to exhibit the sort of morphological changes that occur in natural, biological creatures. The engineering issues that are associated with these kinds of development are numerous. First, it requires being able to mimic the stages of morphological development that creatures undergo as it ages from infancy to adulthood. This is accomplished by multi-target blending between body types. The transitions between "bodies" is analogous to the aging process a creature would undergo during its life cycle. Of course it should be possible to modulate additional body changes on top of this aging process. For example, a creature's weight will generally fluctuate in response to environmental factors during its lifetime. The engineering of a virtual morphology system should supports these changes.

The "virtual development" described above should not only manifest itself in the creature's appearance, but also in the quality of its motion. An extremely heavy creature should move differently than a thin creature of the same age; likewise, an aged creature should move differently than a young creature. A virtual morphology system should be able to handle this manner of motion blending.

Beyond these biologically inspired functions, additional morphological functionality is often valuable to character designers. The more expressive a character is, the better it is able to convey its intentions to an observer [11]. This expressiveness is sometimes not accessible through joint animations, the standard mechanism for character animation, but requires mesh deformation. The bulging of eyeballs when surprised or other exaggerated deformations inspired by traditional animation should be accessible through a general morphology system [37].

Finally, a functioning morphology system needs to be integrated into a higher level behavior architecture. The work done in this thesis integrates with the behavioral research of the Synthetic Characters Research Group at the MIT Media Lab to form a unified architecture for behavior and morphology.

1.2 The Synthetic Characters Research Group

The core research of the Synthetic Characters Group is in designing intelligent systems that draw their inspiration from the field of animal behavior. Referencing the large body of scholarly work in this area, the group endeavors to build computational models of learning and behavior that mimic the everyday commonsense behavior found in small, intelligent animals such as dogs. The group takes this inspiration and uses it to create intelligent, autonomous creatures that are situated in virtual environments. In the end, the goal is to create animated characters that, through appropriate behavior, appear "alive" to users who interact with them.

Fig 1-1 Duncan the Highland Terrier in Trial by Eire

1.2.1 Duncan

Over the past year, the Synthetic Characters Group has engineered a new behavior architecture drawing on the lessons learned from the previous five years of research [6]. This incarnation of the behavior system, named C4, will be describe in Section 2.2. This behavior system is used to drive the creatures in the Synthetic Characters' most recent installation *Sheep|Dog: Trial By Eire*. In this installation the user plays the role of a highland shepherd working his trusty terrier through a virtual sheep herding trial. The installation's lead character, Duncan the Highland Terrier, showcases many of the systems reactive capabilities from responding to voice commands to sheep avoidance. A companion installation features Duncan in a training scenario where the shepherd is training him to perform actions using a reinforcement learning technique called *clicker training*, a proven method for training real dogs [31]. This second installation showcases the ability of the C4 system to learn and adapt to environmental stimuli. Neither of the installations, however, maintains a model of long-term learning and persistence that would facilitate the effective employment of a morphology system, though this is an active area of research within the group.

1.2.2 Wolves

A portion of the group's current research is focused on issues of development and learning within a social hierarchy, like those found in a wolf pack. It is within this developmental context that the idea of character morphology has immediate applications. The wolf pack installation that is currently under development for SIGGRAPH 2001 features six wolves, three adults and three pups, whose behavior is influence by vocalizations (howl, whine, etc.) from the user [38]. How the user directs the pups to interact with the other wolves determines how the pup develops within the pack's hierarchy. Over the course of a ten to twelve minute interaction, the pups develop into adult wolves and take their place in the social

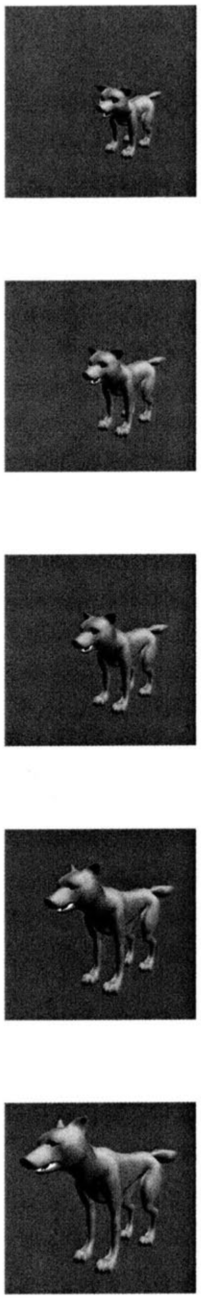


Fig 1-2 Wolf development. models courtesy of Adolph Wong

hierarchy. This behavior-driven development is facilitated by the Morphology System. For this installation it has been fully integrated into C4 and manages the blending of the wolves meshes, skeletons, and animations as they grow into adulthood.

The Morphology System gives virtual characters the ability to develop as described above. This is only one feature of its functionality. Through other mechanisms the system also gives a character the ability to squash and stretch, layer animated facial expressions and body deformations, blend material properties and more.

Before covering the engineering of the Morphology System, a little background is necessary to understand the nature of the work done in this thesis. First, I will cover the general problems associated with polyhedral morphing and relevant work done in this field. This will be followed by a brief summary of related works that deal with aspects of character morphology.

1.3 Related Work

1.3.1 Morphing in general

There is a considerable amount of literature that addresses the complex problem of three-dimensional, polyhedral morphing. The two problems that are associated with morphing between three-dimensional meshes are first finding the correspondence between the two meshes and then interpolating between the two. Of the two problems the first is the more difficult and has been the focus of much of the literature in this area [9][20][40]. This problem is made difficult when the two meshes are not topologically equivalent (not homeomorphic), meaning that either the vertex-counts of the meshes are different or that the geometries are of different genus (i.e. how many holes it has through it). The correspondence problem is generally addressed by putting the two meshes through an preprocessing stage which modifies the meshes to create correspondence [9][20]. This process is involved and often requires considerable user control to achieve the desired results.

The work in this thesis does not attempt to solve the correspondence problem, still an active area of research within the computer graphics community, but circumvents the problem through intelligent character design. The techniques for designing homeomorphic character meshes are described in Chapter 4. Using these techniques it is quite easy to create homeomorphic meshes for blending, even when the shapes are vastly different (Fig 1-3). The only limitation of this approach is the problem of changing genus. While it is be possible to design around the problem by making a hole have zero volume in one of the morph targets, this process is difficult and the results are certainly not satisfactory because the mesh normals generally reveal the discontinuity in the mesh. However, it is safely assumed in this thesis that the morphological changes a character will go through in its life-cycle will not require it to suddenly develop a hole in its body.

Once a correspondence has been established between geometries, the morphing problem is reduced to an interpolation of vertex characteristics. There have been many different schemes proposed for shape interpolation [4][9][23][36], but here simple linear interpolation produces satisfactory results. The vertex attributes that are generally available for interpolation include position, color, UV mapping coordinate, and in the case of "skinned" meshes, bone influence weights (see section 3.2 for details on skinned geometry). Which attributes to interpolate depends on the application and the desired performance, generally the fewer the attributes to interpolate, the faster the blending.

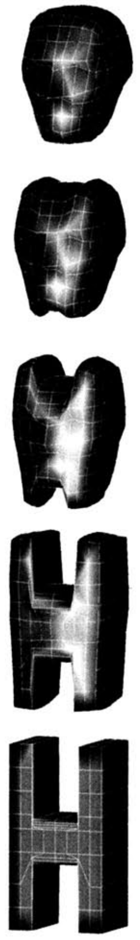


Fig 1-3 Topological equivalence across disparate shapes.

1.3.2 Animation retargeting

The average virtual creature is made up of an external body and an internal skeleton that drives the motion of the creature and the deformation of its skin. When characters morph, not only does the mesh blend, but most often the skeleton is required to blend as well. If the skeleton changes proportions through morphing, animations that were tailored to the first skeleton will not produce plausible motion on the blended skeleton. For example, naively applying an adult's walk animation to a child will produce an awkward, broken animation with an unnatural gait and sliding feet. Retargeting an animation from one skeleton to another takes into account these changes in skeletal structure and attempts to preserve the qualities of the original motion across these changes.

Methods for retargeting animations have been an active area of research in recent years. The work of Gleicher [15], retargets motion by specifying specific characteristics of the motion as constraints (e.g. foot plants during a walk cycle or the hand position of a character reaching for an object), and attempting to maintain them across different skeletons. These constraints are used as inputs into a spacetime constraint solver that considers the changes in limb proportion and the frequency of the original motion and then calculates the modifications needed to make to the original animation data to fit it to the new skeleton [15]. In practice this is an effective way to efficiently reuse animations, thereby limiting the workload on an animator or motion capture specialists. This approach however, by definition, does not produce qualitatively different animations from one skeleton to the next; a baby's walk cycle retargeted for use on an adult, for example, will produce an animation that looks like an adult trying to walk like a baby.

A different approach is taken by Hodgins and Pollard in [16]. The system they describe produces motion, not with keyframed animations or motion capture, but by using a physically based control system that takes into account limb mass and moments of inertia and then uses these inputs to calculate the forces and torques required to produce motion. The system is able to adapt a tuned control system to work on a character of different proportions, effectively retargeting the motion to the new character. Because the retargeting is a function of the physical properties of the new body, the animations do have a qualitatively different appearance. The drawback of this approach, however, is that it replaces the expressive and emotive qualities of keyframed animation with purely functional motion.

While both of these techniques produce compelling results, neither is immediately appropriate for use in this work. The first does not produce animations with independently unique qualities (e.g. a baby-like walk *and* an adult-like walk*)*, and the second takes the animation out of the hand-animated, sample-based realm that the Synthetic Characters Group considers so important for character expression.

1.3.3 Artificial Life and Other Morphing Creatures

1.3.3.1 Karl Sims - Evolving Virtual Creature

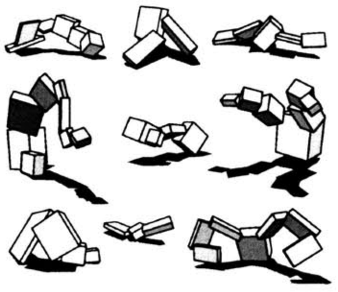


Fig 1-4 Sims's evolved creatures [35]

This seminal work published by Sims in 1995 examines the idea of evolving creature morphology and behavior using genetic algorithms and a simulated physics environment [35]. The "genetics" of each creature dictate its morphology, specifying the size of limbs, how they interconnect, and the degrees of freedom available at each joint. The genes also dictate the creature's motor and sensory controls. To evolve the creatures, each generation is subjected to a *fitness test*, for example, how far they can locomote in 10 seconds. The top performing creatures are allowed to reproduce and pass on their genes to following generations. This mixing of the gene pool along with random mutations, creates new generation of morphologically unique creatures. Over many, many generations, these creatures evolve bodies that are ideally suited to their environment (as measured by the fitness test).

While eminently compelling as an example of functional morphology in virtual creatures, the problems addressed by Sims ultimately lie outside of the scope of this thesis. More immediately relevant is the work LionHead Studios has done in the recently released *Black and White*.

1.3.3.2 Black and White

Black and White, a computer game by LionHead Studios, a U.K. based computer entertainment company, develops the idea of creature morphology along similar lines as this thesis [22]. In the game, creatures' behavior systems drive morphological changes in their appearance. The premise of the game is that the user is a disembodied deity with dominion over a small population of villagers. The users intentions are conveyed primarily through direct intervention in worldly affairs, but his will is indirectly manifested through the behavior of his "pet creature." This creature observes and learns from the behavior of his deity. If the user is evil and unjust, the creature will learn cruelty; likewise if the user is just, the creature will learn to be equitable. These changes in the creature's disposition manifest themselves not only in its behavior, but also through a slow morphing of the creature's body and material properties..

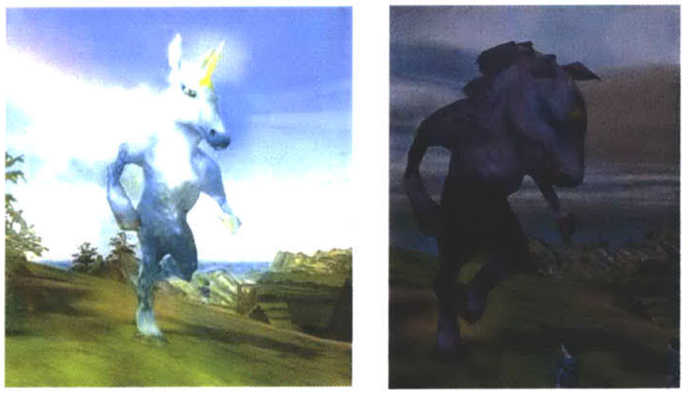


Fig 1-5 A morphable creature from LionHead Studios' *Black and White* [22] *all rights reserved*

Chapter 2
C4 Architecture

2.1 Introduction

The Morphology System was developed from the start to be integrated into the Synthetic Character's behavior architecture, the current revision called C4. C4 is latest in a series of brain architectures developed by the Synthetic Characters research group. This architecture informed many of the design decisions that were made regarding the functionality of the Morphology System and provided the scaffolding for its development. In order to understand these design decisions and how the Morphology System integrates into C4, it is first necessary to understand a bit about C4. This section provides an overview of the core functionality of the C4 architecture and how it relates to the Morphology System. It does not, however, cover the details or the extended functionality of this powerful system. For these details I refer the reader to the recent work of Isla et al [17].

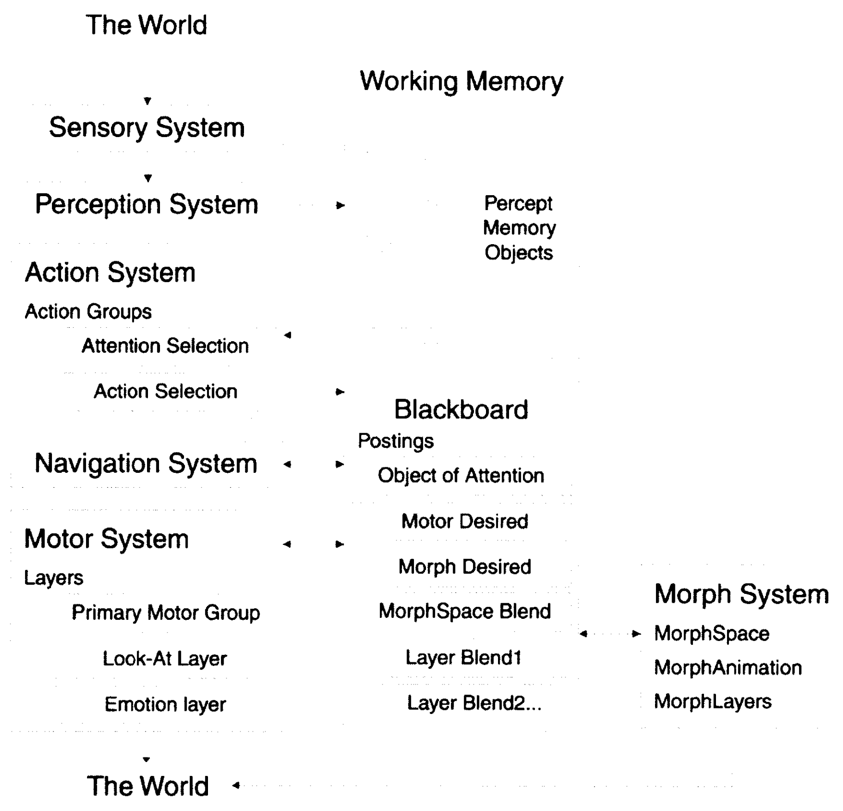


Fig 2-1 C4 system diagram

2.2 C4 – a layered brain architecture

The Morphology System is a modular system that is integrated into the C4 architecture as shown in Fig 2-1. Given the interconnections of the architecture, every system, either directly or indirectly, interfaces with the Morphology System.

2.2.1 Sensing and Perception

The first two layers in the C4 architecture, the Sensory and Perception Systems, indirectly support the Morphology System by situating the creature in the world and giving it the ability to perceive stimuli that might trigger changes in morphology.

The Sensory System is the top layer of the creature's brain and is responsible for collecting and filtering the events that occur in the world. After the Sensory System has filtered the data, it is handed off to the Perception System for processing.

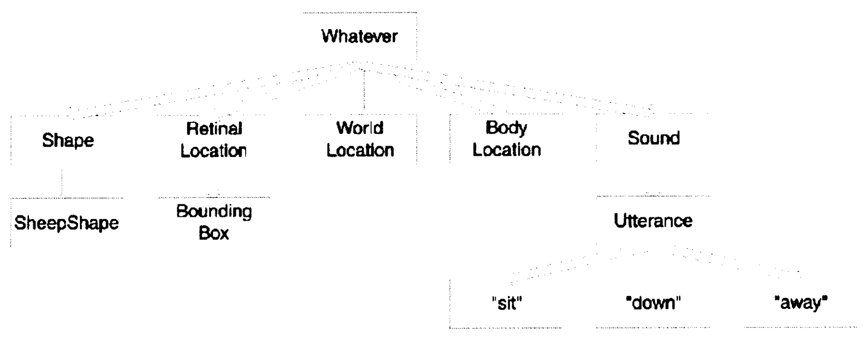


Fig 2-2 A simple Percept Tree

The Perception System exists in a form called the Percept Tree, which is a hierarchical arrangement of perceptions, or "percepts," in order of increasing specificity (Fig 2-2). A percept represents "an atomic classification and data extraction unit that models some aspect of the sensory information," and is passed into the Perception System by the Sensory System [17]. For example, if a creature's Sensory System sees a sheep in the environment, it will pass a SheepShaped percept into the Percept Tree. This percept will be evaluated at each level of the Percept Tree for a match. If it does match, it will be passed down the branch and evaluated by each of that percept's children for a match, and so on. This SheepShape percept will work its way down the tree and eventually activate the SheepShape percept seen in Fig 2-2

2.2.2 Action System

The Perception System interfaces with the Action System through a "percept repository" call Working Memory. The Action System takes the output of the Perception System and is able to orchestrates direct changes in the Morphology System. The Action System is the core of the C4 architecture and is responsible for processing the percepts, selecting the appropriate actions based on these inputs, and then passing messages along to the Motor and Morphology Systems to execute these actions.

The building block of the Action System is structure called the ActionTuple. The ActionTuple is a small computational structure that, when active, determines the current action of the creature. Every ActionTuple contains information that specifies what its action is, when the action should become active, how long it should be active for, what the target of the action should be (if any), and how valuable this action is to do. This functionality is provided by these five subcomponents:

1. *Action:* This is the component that determines what action will take place if the ActionTuple becomes active. The action will generally modify the contents of the internal blackboard (section 2.2.3), the MOTOR_DESIRED or MORPH_DESIRED entries for example, to get a lower-level system like the Motor System or the Morphology System to begin executing the action.
2. *TriggerContext:* the TriggerContext is the component that determines what conditions are necessary for the action to become available for activation. This component most often uses percept information to trigger the ActionTuple. In an "eat" ActionTuple for example, the TriggerContext could be the presence of a FoodShape percept. Thus when food is present, the trigger returns positive and the "eat" ActionTuple becomes active. More complex combinations of triggers are also possible. Using the same "eat" example, a trigger could be built where the creature would eat only if there was food present and he was told to eat. This trigger would require a FoodShape percept as well as an EatCommand percept to become active.
3. *DoUntilContext*: This context determines how long the action should be active, essentially determining the continuing relevance of the action. It can be specified as a constant length of time, a condition that waits until the original trigger context is no longer true, or other customized ending-conditions.
4. *ObjectContext:* This context represents the object that the action will act upon, the "direct object" of the verb. In the "eat food" example, the food would be the ObjectContext. This context is posted to the internal blackboard as the OBJECT_OF_ATTENTION and becomes available to other systems. The Motor System, for example, might be trying to look at whatever the object of attention is and will query the blackboard for this information. Since not all action are "done to" something, this component is not always used.
5. *Intrinsic Value*: This value represents how inherently "good" an action is to do. This value is important in selecting between multiple actions whose triggers are active. The action that has a higher intrinsic value is reliably selected over the other, lesser value actions. This value may be modified by learning and reinforcement.

An example will help clarify how the Action System and the ActionTuple interface with the functionality of the Morphology System. We will continue with the "eat food" example introduced above. The ActionTuple that was being constructed has a *TriggerContext* that activates in the presence of a FoodShape percept (i.e. when there is food present, I can eat it). When this tuple gets activated it begins to perform the *Actions* that are associated with it. The first obvious *Action* to include is an "eat food" motor action. This executes the actual animation of the creature eating the food. But because an ActionTuple can have more than one action, we can also include morph actions. We will include two. First, when the creature starts to eat, we will synchronize a MorphAnimation with the "eat" animation. This MorphAnimation will bulge the creature's cheeks out to the sides while it is chewing, increasing the expressiveness of the action. When the creature finishes chewing, the MorphAnimation will transition the face back to normal. The second morph action will slightly increment the creature's weight as a result of eating the food. This change in weight will be passed to the Morphology System via the internal BlackBoard and the creature's body will be morphed to reflect this change in weight.

The ActionTuple structure is extremely flexible and can accommodate a wide range of custom actions and trigger. By appropriately designing these actions and triggers, the Action System is able to access the full functionality of the Morphology System.

2.2.3 The Internal BlackBoard

Within a single creature there is important information that needs to be passed between systems at every clock cycle. This information interchange in facilitated by a shared blackboard. To enforce modularity, systems do not communicate directly with other systems within the creature. Instead, systems post pertinent information to the internal blackboard along with the time that it was posted and a token identifying what kind of information it is. Once a system writes to the internal blackboard, any other system within the creature can poll it for the latest information. As an example, when the Action System is finished updating it will post an entry to the internal blackboard under the token MORPH_DESIRED, which represents the morphAnimation that the Action System has determined should be active. When it is time for the Morphology System to update, it will query the blackboard for the MORPH_DESIRED token and base its update on this posting. It doesn't care who posted it, only that it is the most recent morph request in the blackboard. This pathway is utilized extensively by the Morphology System for receiving blending information and the desired MorphAnimations.

2.3 Motor Systems

The Motor System is the layer of the C4 architecture that is responsible for animating the virtual creatures. The inner workings of the Morphology System are intimately connected with the functionality of the Motor System. When the Morphology System modifies a creature's structure, the Motor System needs to adapt its animations to reflect these changes. Because the functionality of the Motor System is so important to character morphology, it will be covered in considerable detail.

2.3.1 Background

The Motor System is the layer of the C4 architecture that is responsible for animating the virtual creature. A creature's motor actions manifest themselves as animations played out on the virtual skeleton of the creature. These animations can be created in numerous ways that include motion capture, hand animation, or physical simulation. In the Synthetic Characters Group we believe that creature that maintain complex internal state must be able to convey this information to the observer through its appearance and quality of motion. To meet this end we take inspiration from the expressiveness and dynamic range of classical animation. We believe that a talented animator can produce far more expressive motion than could ever be achieved with procedural or even motion captured animation. For this reason, our Motor Systems are based around sample animations that are always handcrafted by animators.

Considerable research has recently been done on the use of sample animations as input into real-time graphic systems. The work of Rose et al [34] serves as the foundation for much of the motor system development in the Synthetic Characters Group. This work introduced the idea of *Verbs and Adverbs* (later expanded on by Rose in [32]). This technique uses radial basis functions (RBF) to blend between similar animations (the verb) each with a unique emotional characteristics (the adverb). Thus, given multiple input animations each with different characteristics, an RBF blend gives a single output animation that contains a mix of the characteristics from the input animations. This functionality is extremely important to character morphology because it allows the characteristic motions from different stages of development to be blended as a character develops. For example, an animator may create two sample animations, a skinny-walk and a heavy-walk. In this example the verb "walk" could be said to live in the "skinny-heavy" adverb space. Using the techniques proposed by Rose, these animations are blended in real-time giving the creature access to an entire range of animations between skinny and heavy. The benefit of this approach is that for relatively few sample animations a vast set of animations become available to the Morphology System.

Rose also introduces the idea of the VerbGraph in [34]. The VerbGraph is

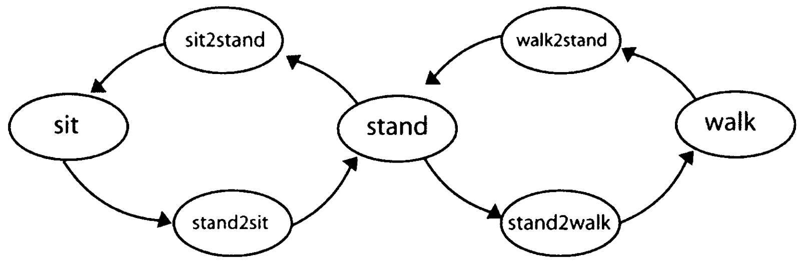


Fig 2-3 A simple VerbGraph

a system for properly sequencing short animation segments for output to the creature. It consists of a directed graph of "verb" nodes, each representing an animation, in which the edges of the graph represent all the allowable transitions into and out of that particular node. This graph, in essence, governs the transitions between the different animations that are available to the creature. Take a creature that is lying down, for example. This creature cannot go from a lying down animation straight into a running animation without an embarrassing discontinuity. A properly constructed VerbGraph demands certain transitions to take place before the creature could run. The graph would require the creature to transition from the lie down animation to stand, then stand to walk, and finally walk to run. These transitions could be hand animated and inserted as nodes into the graph or they could be produced automatically [33]. Fig 2-3 shows an example of a simple VerbGraph. While not directly interfaced with the Morphology System, the functionality of the VerbGraph is important for believable, real-time character animation.

Another feature that is indispensable for creating compelling characters is *animation layering*. As described by Perlin in [28], a motor system should have the ability to override certain degrees of freedom (DOFs) in a character's skeleton to play higher priority animations while the initial animation continues to play on the rest of the skeleton. Take for example a character who is walking and wants to wave his arm. Initially the walk animation controls the DOFs in the arms, but when the higher priority "wave" animation is requested, it takes control of the arm DOFs, executes the wave animation and then returns control of the arm to the walk animation. The Morphology System can synchronize morph animation with these animation layers to enhance the expressiveness of a character's animation.

There are currently two motor systems available in the C4 architecture. The first, called simply the C4 Motor System, implements the blending of Rose and the layering of Perlin; the second, called Posegraph is an area of continuing research within the group that extends the standard set of motor functionality into the realm of motor learning. Both motor systems can be used interchangeably with the Morphology System.

2.3.2 C4 Motor System

The C4 Motor System is a basic implementation of the ideas presented above. It allows for multidimensional verb-adverb blending and has a simple implementation of the VerbGraph. It also supports the type of layering proposed by Perlin. It does not however support the real-time generation of transitions.

The system integrates with the Morphology System and the rest of the Character's architecture through the internal blackboard. At every update the Action System passes the desired motor skill into the creature's internal blackboard. During the motor system's update, it queries the MOTOR_DESIRED and MOTOR_ADVERB slots in the blackboard to determine what animation should become active and what blend (adverb setting) that animation should have. The Motor System takes the desired motor skill and compares it to what is currently running. If the two match, the animation continues to run. If they are different, the Motor System begins a traversal of the VerbGraph finding the shortest path to the desired node, and then begins to execute the sequence of animations required to complete this traversal.

2.3.3 Posegraph

The Posegraph motor system represents the vanguard of motor research within the group. The system developed by Downie [12], extends the functionality of the C4 motor system by introducing a new representation for animation data. Instead of loading animations in as atomic animations "clips", animations are loaded into the Posegraph system and then disassembled frame by frame into discreet BodyPoses. These BodyPoses are then connected into a complex directed, weighted graph that could be though of as a hyper-detailed VerbGraph. Each BodyPose contains information on the rotation of each joint in the pose, the velocities at each joint, as well as information about what animation it originated from and when. This unique representation of the animation data coupled with a robust underlying architecture enable the Posegraph motor system to do many useful things. Among them, it is able to generate informed automatic transitions that can be saved out, edited, and reused; it is able to generate new animations by spatially aligning relevant BodyPoses; and it is able to constrain end-effector positions, such as foot falls, at runtime. The latter proves especially valuable to the Morphology System as a means of ensuring a character's feet remain on and attached to the ground throughout its skeleton blending. For details on the inner workings of the Posegraph see the recent work of Downie [12].

Beyond these representational differences, the Posegraph motor system integrates with the Morphology System and the overall C4 architecture through the same MOTOR_DESIRED and MOTOR_ADVERB mechanisms described above.

2.3.4 AdverbConverters

Both the C4 Motor System and the Posegraph Motor System rely on a computational device called the *AdverbConverter* to supply valid adverb information to the Motor System. The utility of the AdverbConverter lies is in its ability to take simple, intuitive adverb representations and convert them into the actual blend weight values that the Motor System requires. For example, adverbs are quite often used for navigation. A navigation specific AdverbConverter, called a BearingAdverbConverter, takes a single value between -1 (turn hard left) and 1(turn hard right), and converts it into three blendweights which the Motor System uses to determine how much of each animation, *walk_left, walk_straight, walk_right*, to blend into the output animation. The Morphology System makes extensive use of custom AdverbConverters to interface with the Motor System. It is through this mechanism that a character's morphological changes are reflected in its animations (see Section 3.3.3).

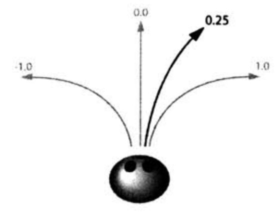


Fig 2-4 Adverb conversion of three animations to produce a directional walk

2.4 Summary

At each clock cycle, every creature in the virtual world refreshes its view of the environment and decide how to respond to changes. The incoming sensory information is first received by the Sensory System. This information is filtered into the Perception System where it is sorted and classified into different percepts. Based on these percepts, the character's Action System updates and decides what is the most appropriate action for the creature take given his current view of the world. The Morphology System is the next to update. During this update, the Morphology System updates all the installed components adjusting the creature's morphology as necessary. When complete, the Motor System gets updated. The Motor System then starts, stops, or updates animations as per the Action System's request, taking into account any changes in the character's morphology. At the end of all these behavior and motor computations, the graphics layer gets updated and renders the characters on the screen.

Chapter 3
The Morphology System

3.1 Introduction

The Morphology System is a complete, modular system that has been developed to be easily integrated into the Synthetic Character's behavior architecture. This behavior architecture gives creatures the ability to maintain internal state about their drives, emotional state, beliefs about the world, and even their age. The Morphology System was designed to allow cross wiring between any combination of these parameters and the character's morphology.

When a character is created, a designer authors character specific extensions to the major systems within C4: the Action System, Motor System, Perception System, and the Morphology System. These character specific systems extend the functionality of the general systems by adding the details that are specific to each character, thereby giving it its uniqueness. For example, the Action System of Duncan extends the functionality of ActionSystem.java by adding "Duncan specific" actions.

Similarly, a Morphology System is added to a creature to give it specific control over how and when its body will change. To control these changes, the Morphology System has three main components. They are MorphSpaces, MorphAnimations, and

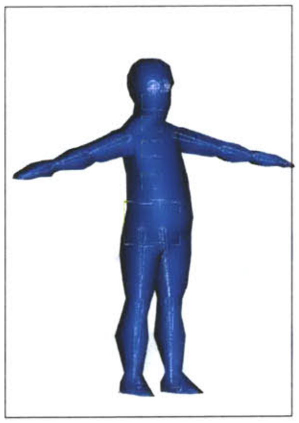


Fig 3-1 A"skinned" character

MorphLayers. These components can be added to the creature's Morphology System independently or in combination, each providing specific functionality.

The MorphSpace is the principal component of the Morphology System and performs the morphing associated with long-term, gradual morphological changes like aging or gaining weight. The second main component, the MorphAnimation, is essentially a sequence of "body" keyframes with user specified transition times that can be played out on demand. The useful characteristic of the MorphAnimation is that it can be integrated into an actionTuple and thus be directly controlled by a creature's action selection. This component is a powerful tool for creating exaggerations that lie outside the range of what is possible within the confines of a MorphSpace. MorphLayers are the final component of the Morphology System. Layers give the character designer the ability to layer additional blends on top of the blending that is already done in the MorphSpace and the MorphAnimations without destroying the previous blending. Layering is useful for local deformations like facial expressions. Before I present the engineering details of these components, an primer in the anatomy of a virtual character is necessary.

3.2 Anatomy of a Virtual Character

Graphically, a character is made up of:

1. Polygonal meshes, which may be only a single continuous mesh or many segmented meshes

2. A skeleton of TranformControllers (TCs) which are the virtual "joints" of the character and contain a default set of positions and rotations that give the skeleton its shape

3. Material properties that contain information on the character's color, specularity, and a texture maps

The exterior of a virtual character is described by its polygonal meshes. There can be few or many meshes that make up a character's body. In a simple character like the BlueMan shown in Fig 3-1, three meshes describe the body, one large mesh that forms the skin and then two meshes for the eyeballs. The amount of detail that can be contained in each mesh is proportional to the number of vertices in the mesh, and of course the more vertices in each mesh the more expensive it is to morph. The BlueMan is quite low resolution, and consequently, is computationally inexpensive to morph. By contrast, the character shown in Fig 3-2 is highly detailed and requires considerably more resources to morph.

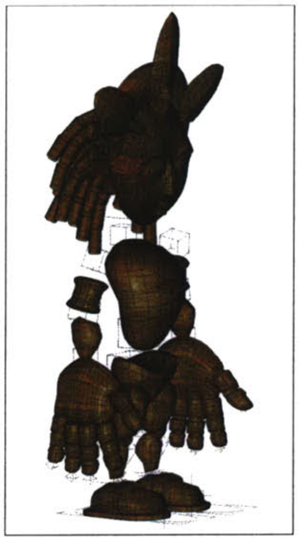


Fig 3-2 A "segmented" character

Generally, a character's meshes can be joined to its skeleton in two ways. The simplest and least computationally expensive method is to directly link discreet meshes to each bone. This type of character is described as "segmented" (Fig 3-2). Of course, few creatures in nature, excepting insects, are segmented. To create a more natural appearance a creature must be "skinned." Skinning lets one mesh be influenced and deformed by the motion of multiple bones, much like the skin of natural creatures.

The BlueMan is an example of a skinned character. He has a single primary mesh whose deformation is influenced by the bones in his skeleton. Skinning produces a generally smooth deformation of the mesh in conjunction with the motion of the skeleton. Which bones influence the deformation of each vertex is determined during the construction of the character, and these values are loaded into a *SkinController* at runtime. It is sufficient to say here that for each vertex, the *SkinController* has a list of all the bones in the skeleton and how much they influence that vertex. These influence are referred to as *bone weights*. For a single vertex in the BlueMan this list of *bone weights* would look something like:

```
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.215626, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.784374, 0.0, 0.0;
```

which in optimized form is simply:

```
Bone07 0.215626
Bone14 0.784374
```

In natural language this says that this vertex, somewhere in the BlueMan's shoulder, is influence by the motion of the bone in the upper arm (78%) and the uppermost spine segment (21%) and not by any of the other bones. For a further details on skinning I refer the reader to [13][25][2][3].

Beneath a character's meshes lives its skeleton. The skeleton is made up of a hierarchy of TransformControllers. A TransformController can be thought of as a joint in the skeleton that contains a *rotation* and a *position*. Because the TransformControllers are arranged hierarchically, modifications at one level propagate down to the children of that controller. For example, if the TC representing the shoulder rotates, this rotation transforms all the shoulder's children in the skeletal hierarchy - the elbow, the wrist, and all the finger joints, effectively rotating the entire arm. Each joint in the hierarchy has its own TransformController that specifies how far away it is from its parent joint (here the shoulder) and any rotations imparted at that particular joint.

Fig 3-3 Detail of segmented meshes

A creature's TC list contains not only these "joint" TCs, but also a Transform Controller for each mesh that is linked to the skeleton (as in a segmented character). These additional TCs tells the graphics system where and in what orientation the meshes should be attach to the skeleton. This hierarchy, starting with the "root" TransformController (generally around the hip area) and extending out to the tips of the fingers and toes, describes the entire skeleton and, together with the linked and skinned meshes, makeups the creature's body.

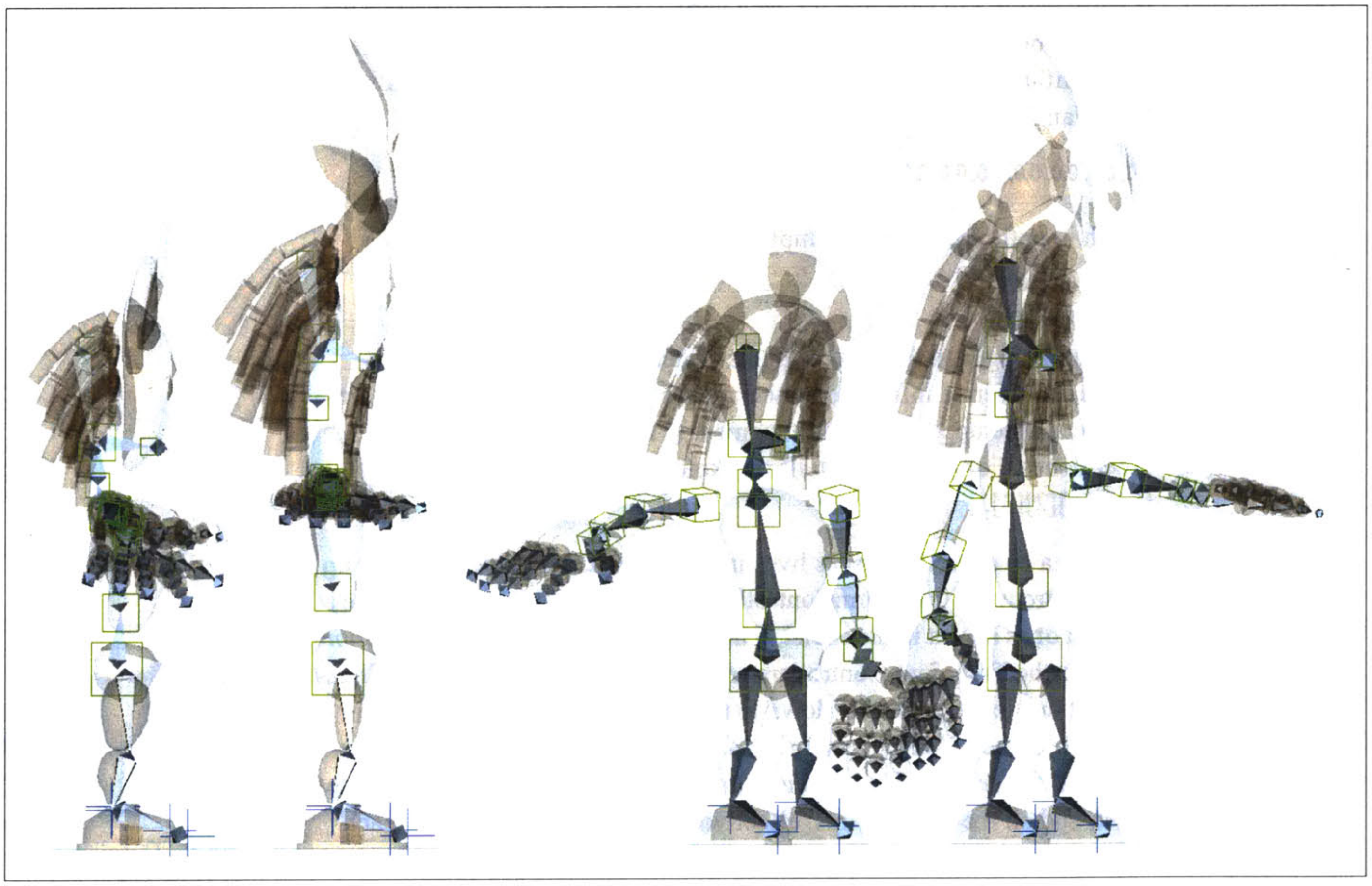


Fig 3-4 Skeletal hierarchy of the Zulu Warrior before and after aging

3.3 The Engineering

3.3.1 MorphTools

The MorphTools are the work-horse of the Morphology System. This class handles all the low-level computation required to morph the meshes and skeletons on the screen. While system components such as the MorphSpace decide how the creature should be morphed, the MorphTools make this happen.

When the Morphology System needs to morph a creature, the skeleton is the first thing to get blended. A call is made to the blendSkeleton() method in the MorphTools. This method performs multi-target blending on the skeleton and takes as arguments: *n* input skeletons, *n* blend weights (one for each input skeleton), and an output skeleton to receive the results of the blending. The positions and rotations of every TransformController in the skeleton are interpolated according to the specified blend values and the result is placed in the output skeleton.

Mathematically, each TransformController in the skeleton encapsulates a three dimensional position vector and a unit quaternion rotation. The position vector is blended using standard linear interpolation. The quaternion rotations are first logged and then interpolated in Euclidean tangent space and then transferred back again using the approach outlined in [19]. In general the blending of TC and vertex characteristics is governed by:

$$v_{interp} = \sum_{i=1}^{n} \beta_i v_{ij} \qquad (1)$$

where v_{interp} represents the output value of an multi-dimensional blend given *n* input values and *n* blendweights (β_i).

Once the skeleton is blended, the meshes are blended using the same set of blend weights. The mechanism for the mesh blending is provided by two methods in the graphic system, they are MultiplyAndSet (MAS) and MultiplyAndAdd (MAD). Both methods operate on a single output mesh and take as arguments a blendweight and an input mesh. The first method, MAS, multiplies each vertex value by the specified blendweight and overwrites the corresponding vertex value in the output mesh, in essence resetting the blend on the output mesh. The second method, MAD, multiplies each vertex by the blendweight and then adds the result to the corresponding value in the output mesh. Here is a simple example to illustrate the use of these methods.

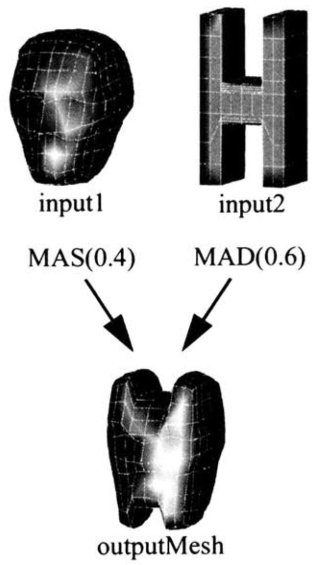


Fig 3-5 Two inputs into a output mesh

```
MagicGeometry outputMesh = getMeTheMeshToDisplay-
    OnTheScreen();
MagicGeometry inputMesh1 = getMeTheFirstMorphTarget();
MagicGeometry inputMesh2 = getMeTheFirstSecondMorphTar-
    get();

someBlendValue1 = getBlendValue();
someBlendValue2 = 1 - someBlendValue1;

//now blend the meshes

outputMesh.MAS (someBlendValue1, inputMesh1);
outputMesh.MAD (someBlendValue2, inputMesh2);

// show me the blended mesh
showOnTheScreen(outputMesh)
```

Given a mesh called outputMesh, the first command, outputMesh.MAS (*someBlendValue1, inputMesh1*), takes the floating point value, someBlendValue1, and multiplies this by every vertex attribute (position, UV, etc.) in inputMesh1, and sets these new vertex values into output-Mesh. The blending action occurs when output-Mesh.MAD(*someBlendValue2, inputMesh2)* is called, adding the product of someBlendValue2 and the vertex attributes of inputMesh2 to output-Mesh. In essence, the outputMesh serves a "vertex container" that will hold whatever is put into it by the MAS and MAD commands, and keep those values until it is modified again.

When specifying the blend values, predictable results are only achieved when the values sum to one, otherwise the blend extends outside of the convex hull of the examples and the character's appearance begins to breakdown. Section 3.3.2 looks briefly at the extrapolation effects that can be achieved by intelligently blending outside this hull.

Blending skinned meshes follows the same procedure except that the SkinController must first be disabled, effectively "turning off" the skin, before the blending can take place. Once the skin is "turned off", the mesh is blended in a neutral reference frame, called the *bind pose.* When this blending is completed, the SkinController rebinds the blended skin to the skeleton.

3.3.2 Nodes

3.3.2.1 MorphNodes

The building blocks of the Morphology System are data structures called MorphNodes. Each node contains all the morphable data within a character's body Fig 3-6. These nodes are used as the inputs into every component of the Morphology System; they define the extents of a MorphSpace, the keyframes in a MorphAnimation and, along with LayerMorphNodes, form the basis for MorphLayers.

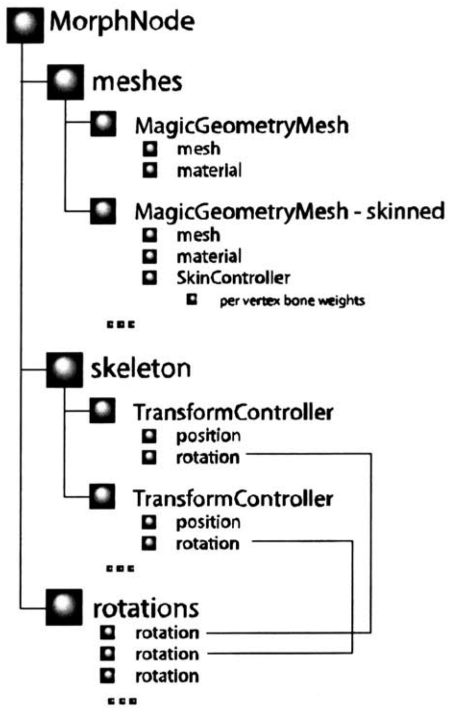


Fig 3-6 Hierarchy of a MorphNode

During the initialization of the Morphology System, all the MorphNodes that will be used by the character are loaded into system memory and maintained there for used as needed. For large, complex models this approach is very memory intensive, especially when more that a few nodes are needed by the Morph System. For example, the character in Fig 3-2 has a total of 24000 polygons, and 190 TransformControllers (including one for each dreadlock) that define his body. Each variation of this body that is loaded in as a MorphNode requires approximately 47 MB of system memory. The node loading mechanisms provides a mechanism for cutting this cost by loading in only the meshes that are going to change, which quite often is not every mesh in the body (the eyes for example may not change size or shape as a character ages, only location). Morphing only the required meshes can results in considerable savings in both memory and computation. In this character, the meshes in both the hands and the feet (8500 polygons total), do not change from young to old. Using only the necessary meshes results in a 30% decrease in memory consumption and a ~3% increase in framerate.

3.3.2.2 LayerMorphNodes

LayerMorphNodes (also called LayerNodes) contain data structures similar to a standard MorphNode except that these nodes contain the results of a subtraction between two MorphNodes. The result is a node that contains information about the relative changes from one body to the next. LayerMorphNodes can be used as inputs for both MorphAnimations and MorphLayers.

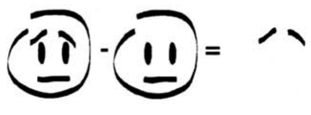


Fig 3-7 Creation of a LayerMorphNode

A LayerNode is constructed by specifying a reference body and a target

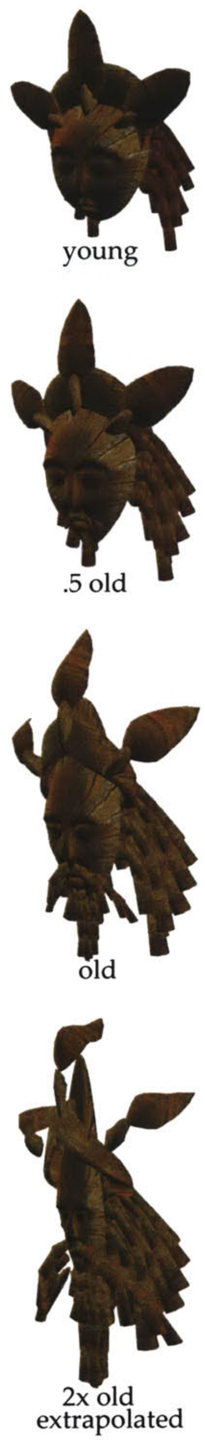


Fig 3-8 Effect of extrapolation

body which become the two operands in the subtraction. The reference body is subtracted from the target body, leaving only the changes that exist between the two. Because this information is relative and not absolute, it allows LayerNode to be added on top of existing blends with out destroying the underlying properties. For example, when a character is aging from young to old, the MorphSpace controls the absolute changes in its appearance. But LayerNodes can be used to add facial expressions. These facial expressions maintain their effect no matter how the underlying MorphSpace changes. These facial expression would be created by using a young, neutral face as the reference node, and then subtracting that from a young, happy face. The result of the subtraction would be how much the vertices around the eyes, mouth and cheeks need to move to make a happy face. Assuming the old face is constructed with correspondence between features (see section 4.1 for detail), these changes in vertex position can be "layered" on top of the arbitrarily aged face with fairly predictable result (see Fig 3-12,13).

Of course this method breaks down in cases where the MorphSpace transitions between vastly different shapes where feature correspondence is not preserved. For example, if the group of vertices that make up the mouth in one mesh are located on the lower chin in the target mesh, the layering of a smile on top of this blend would fail. In most applications where the characters are intelligently designed the method yields effective results.

Another interesting feature of using difference information for layering is that it can be extrapolated beyond the limits defined by the initial samples. The effect of extrapolation is similar to caricature in that it is able to exaggerate the features of a blend beyond what was intended by the designer. Fig 3-8 shows the effect of extrapolating age far beyond the intended blend.

3.3.3 MorphSpaces

The MorphSpace is the primary component of the Morphology System. The MorphSpace allows the user to set up an N-dimensional space that controls a character's morphological development, where each dimension represents an axis of development in the body space. A one-dimensional MorphSpace, for example, would be controlled by a single parameter such as age, and defined by two age samples, one young and one old. The behavior system would modulate the age value and pass it into the blackboard. The Morphology System would query this value at every clock cycle and passes it off to the MorphSpace which would then handle all the body morphing. Similarly, a two dimensional MorphSpace would be defined by four examples (see Fig 3-9), and would require two values from the behavior system to fully specify the blend. In theory, a MorphSpace could be generalized into N-dimensions. In practice though, this space would require blending 2^n meshes, 2^n skeletons and 2^n animations ($3*2^n$ animations for directional locomotion) at every clock cycle. Not only would this be computationally limiting, but the authoring time required to produce all the sample meshes and animations would be quite expensive.

3.3.3.1 Setup

The first step in setting up a MorphSpace is to instantiate an instance of a general MorphSpace. The constructor requires references to the Morphology System that it lives in, the character's internal blackboard (so it knows where to look for updated information from the rest of the behavior system) and the World that the character lives in (for rendering purposes). Once the generic MorphSpace is constructed it is customized by specify a MorphSpaceConverter. The system currently has two MorphSpaceConverters implemented, a MorphSpaceConvertor1D and a MorphSpaceConvertor2D. It is the MorphSpaceConverter's job to take the values the MorphSpace grabs from the blackboard and convert them into the blend values that are readable by the MorphTools. Below is pseudocode from a two-dimensional MorphSpaceConverter that converts two parameter values, valueA and valueB, into a vector of blendweights that gets passed off to the MorphTools.

```
A = 1-valueA;
B = valueA;
C = 1-valueB;
D = valueB;

blend.set(0, A*C);
blend.set(1, B*C);
blend.set(2, A*D);
blend.set(3, B*D);
```

The formulation is simple but useful in that it allows intuitive values, such as age and weight, to be used for A and B. Additional MorphSpaceConverters can be written that define conversions for arbitrary spaces, they need only define a method *convert*(), an return a vector of blend weights that is equal to the number of nodes that define the MorphSpace (i.e. one blend weight for every sample).

Finally, the MorphNodes that define the extents of the space must be added to the space. For a two dimensional MorphSpace, they are specified in the order (0,0) (0,1) (1,0) (1,1) with the node at (0,0) corresponding with the initial body of the character. This ordering is entirely arbitrary and may be customized, but whatever the ordering, it must coincide with the order of the blend weights provided by the MorphSpaceConverter.

An example of a MorphSpace construction:

```
MorphSpace ms = new MorphSpace("Morphspace Name",
   myMorphSystem, CreaturesWorkingmemory, theWorld);
ms.setWhenToUpdate (UPDATE_ALWAYS)
ms.addMorphSpaceConverter(new MorphSpaceConverter2D());

ms.addNode(baby);
ms.addNode(baby);
ms.addNode(skinny);
ms.addNode(fat);
myMorphSystem.addMorphSpace(ms);
```

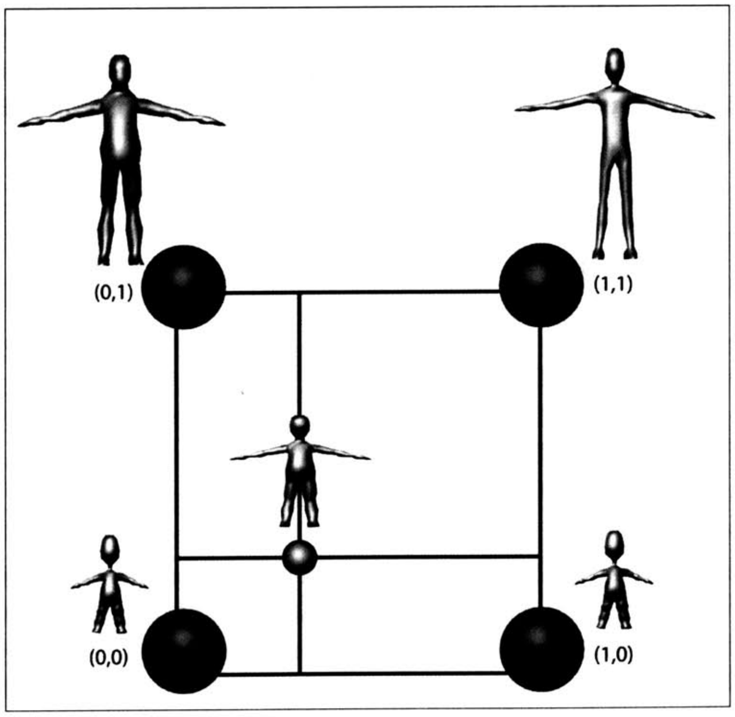


Fig 3-9 A two-dimensional MorphSpace construction

3.3.3.2 Runtime

During the life cycle of a character, the Morphology System is updated at every tick of the world clock. What happens to the MorphSpace during these updates? First the MorphSpace polls the internal blackboard for updates to a vector of blendweights identified by the token "MORPH_BLEND". Once this value is extracted, if the MorphSpace is setup to blend only on changes, it looks for differences in the current values and those from the previous update. If something changed, the creature got a little heavier for example, the values are passed to the MorphSpaceConverter and are modified into a form that is meaningful to the morphing tools. The MorphSpace then makes a call to MorphTools.blendBodies(), and gives this function the current body that is displayed on the screen as the container for the blend, the nodes in the MorphSpace as the inputs to the blend, and the vector or blendweights returned by the MorphSpaceConverter.

Because the MorphSpace produces structural changes in the character's body, it is important to have the quality of the character's animations change to reflect is current disposition in the MorphSpace. A heavy character should not walk like a thin character, likewise a young character should not move like an old character. To effect these changes, a series of AdverbConverters have been developed to synchronize a blended set of animations with the character's current location in MorphSpace. The concept of the AdverbConverter was introduced in section 2.3. In this construction we have the AdverbConverters poll the internal blackboard for the value, MORPH_BLEND, that the MorphSpaceConverter uses. The adverb conversion process is similar to the MorphSpace conversion process described above. The results of the adverb conversion are fed into the Motor System which does the final motion blending. Four AdverbConverters were implemented to work in conjunction with the MorphSpaces.

For a one-dimensional MorphSpace:

1. Adverbconverter1DMS – blends two animations, one for each sample in the MorphSpace
2. BearingAdverbconverter1DMS – blends six animations, three directional animations for each node in the MorphSpace.

For a two-dimensional MorphSpace:

1. AdverbConverter2DMS – blends four sample animations, one for each node in the MorphSpace.

2. BearingAdverbConverter2DMS – blends twelve sample animations, three directional animations for each node in the MorphSpace

In practice the combination of blending meshes and skeletons in conjunction with animation blending produces very compelling results. The integration of the two elements presents a believability to the developmental changes that the character undergoes. I will now describe the functionality of the MorphAnimation.

3.3.4 MorphAnimations

As introduced briefly at the beginning of this chapter, MorphAnimations give the character designer the ability to playout sequences of morph blends with precise control over the length of the transitions. A key feature of the MorphAnimation is its that it can be integrated into the actionTuple in the Action System. This puts command of MorphAnimations under direct control of a creature's action selection. Because Morph and MotorAnimations can be included as actions in the same actionTuple, the designer has the ability to perfectly synchronize morph animations with a motor animations. This coupling can be used for any number of expressive effects, limited only by the designer's imagination.

3.3.4.1 Setup

A MorphAnimation is set up by first creating a new instance of the class MorphAnimation and then specifying the nodes (either MorphNodes or LayerMorphNodes) and the transition times between nodes. An example of this setup is shown below:

```
MorphAnimation testMorph = new MorphAnimation();

testMorph.addNode(80, firstNode, theWorld);
testMorph.addNode(50, secondNode, theWorld);
testMorph.addNode(80, backToStart, theWorld);

//Create my main morph action group
mag = new MorphActionGroup("MyMag", internalBlackboard);
mag.addSkill(makeMeAMorphSkill("MORPH", testMorph));
morphSystem.addMorphAction(mag);
```

In lines 2-4 above, the MorphNodes are added to the animation with inte-

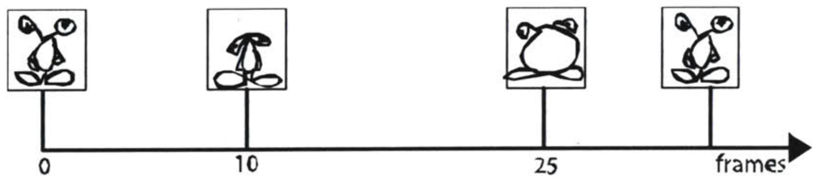


Fig 3-10 A multi-node MorphAnimation

ger transition lengths, specified in frames. The overall animation will be 210 frames long, transitioning over the first 80 frames into the *firstNode*, then to the *secondNode,* and finally back to the point of departure.

The second half of the code above requires explanation. First, a MorphActionGroup is a class that holds all the morph "skills" (structures that encapsulate the MorphAnimations) and makes them available to the Action System. The MorphActionGroup gets updated at every tick and looks at the internal blackboard to see what, if any, morph skills are desired by the Action System. If there is a morph skill in the MorphActionGroup that matches what is being requested by the Action System, the MorphActionGroup activates the skill and the MorphAnimation begins to execute.

Once the MorphAnimation is activated, the MorphActionGroup hands control over the playback of the MorphAnimation to a class called MorphActionAnimation. This class is responsible for tracking the frame count of the animation, transitioning between nodes in the animation, setting the appropriate blend weights for a given point within the animation, and finally calling MorphTools.blendBodies().

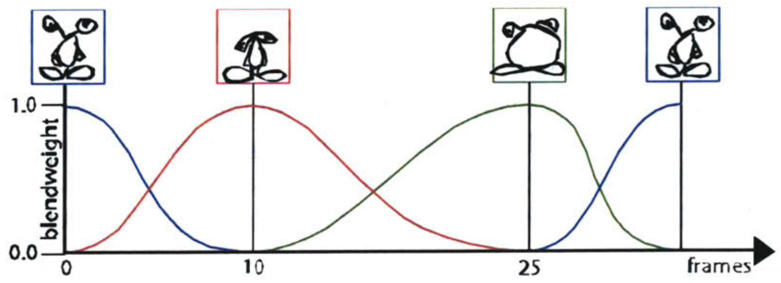


Fig 3-11 Using piecewise cosine basis functions to create morph transitions

For MorphAnimations, the on-screen body node is always used as the container (the output) for the blend of two inputs. When the animation starts, the blend takes the on-screen body from its current representation to that of the first node in the node list. Once this first "morph segment" is completed the MorphActionAnimation looks for another node in the animation. If there is another node, it puts the current node in as the first input to a new blend and sets this next node as the second input to the blend. This cycle continues until there are no more nodes, at which point MorphActionAnimation deactivates.

One important issue that MorphActionAnimation must address is how to smoothly transition between multi-node animations. A suitable blending scheme needs to minimize discontinuities at node transition points. The system currently blends with simple cosine basis functions as shown in 3.3.5. This scheme is effective in that it provides C^1 continuity between nodes, and requires only ever blending two nodes at a time. However, given the representation, the first derivative always goes to zero at the sample points, which is often not the desired behavior. A full radial basis function solution could be implement that would smooth the transitions across samples, but would require blending more than two nodes at a time significatly increasing computation [30].

3.3.5 MorphLayers

MorphLayers are the final component that can be installed into a character's Morphology System. MorphLayers are defined by a single Layer-MorphNode and are controlled by a MorphLayerManager which receives updated layer values from the internal blackboard and blends based on these values. MorphLayers are able to preserve the characteristics of the underlying MorphSpace blends while adding their own unique characteristics.

As one would expect, there is additional computational cost associated with adding MorphLayers to a character. The MorphLayerManager is required to re-blend an individual layer when it changes, but must also recompute all layers whenever the underlying MorphSpace (or MorphAnimation) updates its blend, effectively overwriting all the layering information. So, for a MorphSpace that is set to update at every clock cycle, each MorphLayer must also recompute its blend every clock cycle.

Luckily, morphing layers can be a fairly efficient process. First, only meshes that have layering information on them need be blended. For the Zulu Warrior in Fig 3-12, only the actual face mesh has layering information. Considerable saving are had when layering calculations are done only on this mesh. Similarly, if a layer does not involve any skeletal modifications, it is not necessary to blend the skeleton when updating the layer. These two optimizations produce a 10% savings in computation when running two layers on this character.

Additionally, MorphLayers require only a single MAD command for each mesh (by contrast, each mesh in a MorphSpace blend requires a MAS call, and at least one MAD call, depending on the dimension of the MorphSpace).

One final optimization is achieved by looking at the individual vertex information before it is blended. Ideally, only the vertices that change position (or UVs, color, etc.) would need to be blended, saving computation by ignoring all the zero values that resulted from the initial subtraction that created the layerNode (the information that doesn't change from the reference node to the target node results in zero values). However, there is no guarantee that the vertices that changed live in one contiguous block of memory, and chances are they are scattered throughout memory. This prevents us from isolating and iterating through a single block of non-zero vertices. Short of being able to do this, optimization is had by checking each vertex for non-zero values, and if it is non-zero, then do the multiply operation on it. This optimization trades off the computational expense of a *compare* operation with that of a *multiply* operation across thousands of vertices, producing a minimal but still relevant 1% increase in framerate when tested on the expression layers in Fig 3-12,14.

3.3.5.1 MorphLayer Applications

The power of MorphLayers lies in their ability to update independently of the underlying MorphSpace blends. This has a few interesting applications. The most common application of layering is for facial expression [24]. This effect is shown in Fig 3-12,13. Here the layers were created by differencing a young, neutral face with a young, expressive face. The subtraction produces a single set of layers that provide a good approximation of the original expression across any age in the Morph Space. The figures show the layers being applied to both the young head, which they would of course work on, and on the old head. Because the old head was designed with feature correspondence (Section 4.1), the layers provide an excellent approximation of the original emotion. It is also interesting to note that the full application of two layers, happy and angry, produces an emergent third emotion, mischievous.

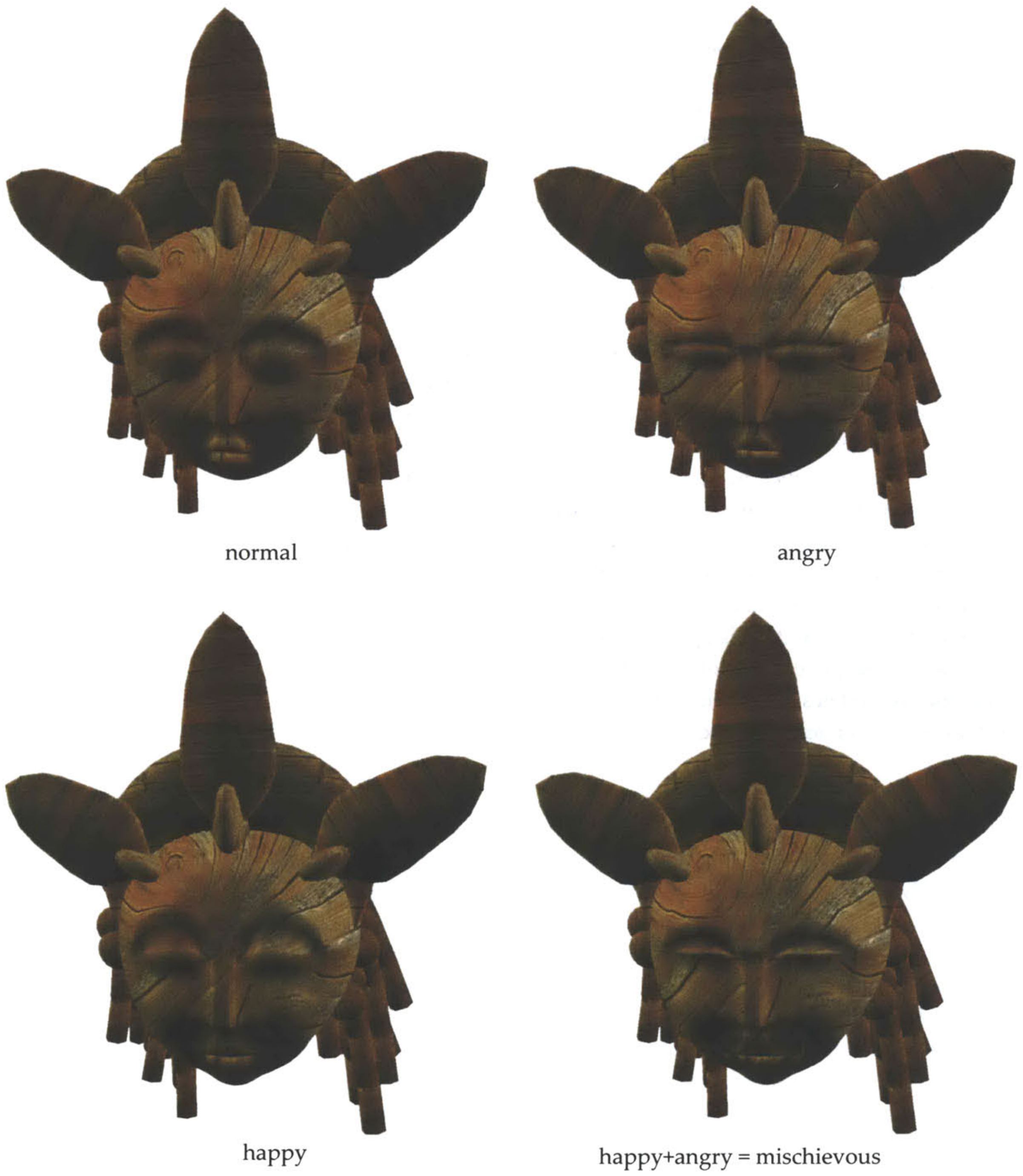


Fig 3-12 The effect of layering on a young face

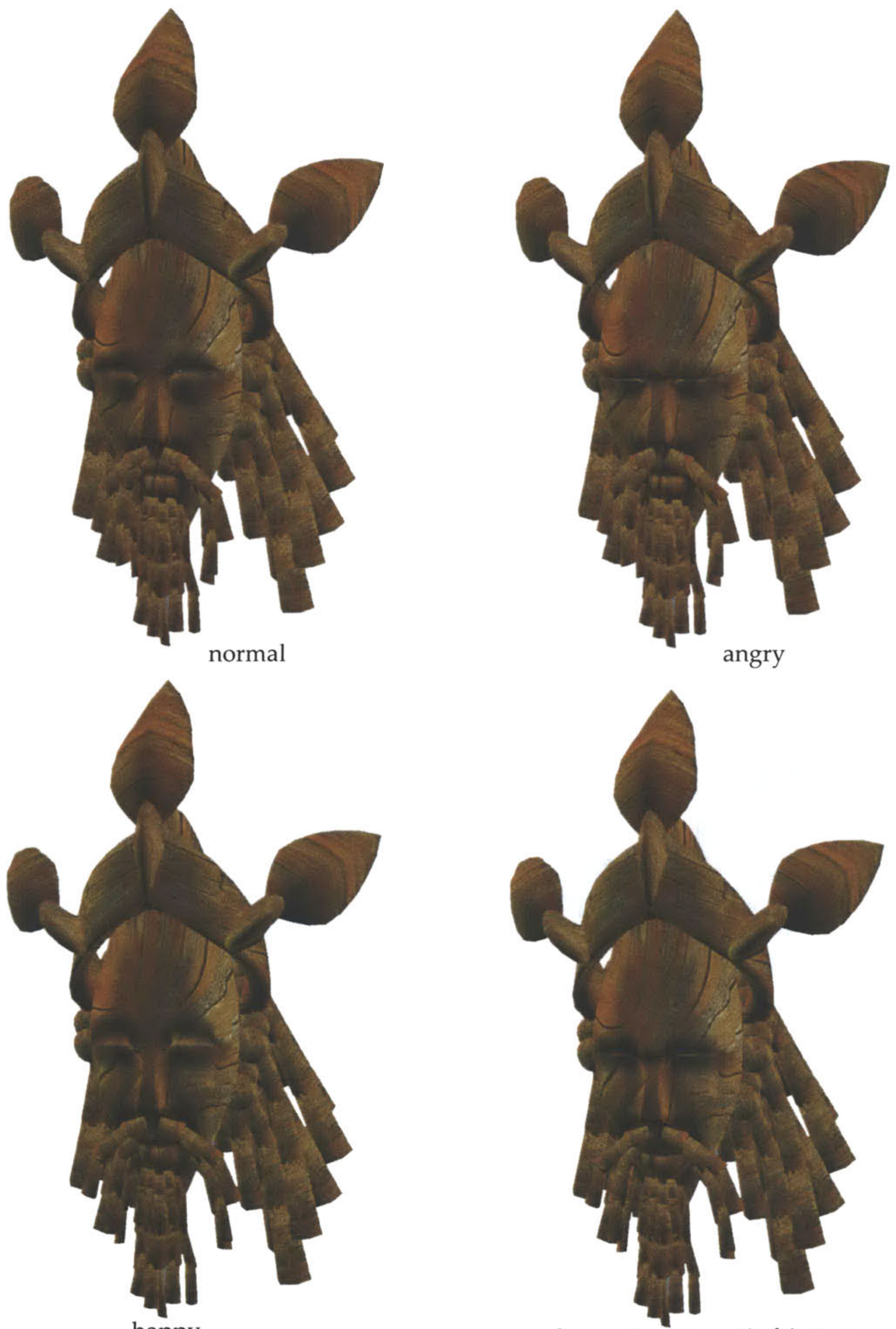
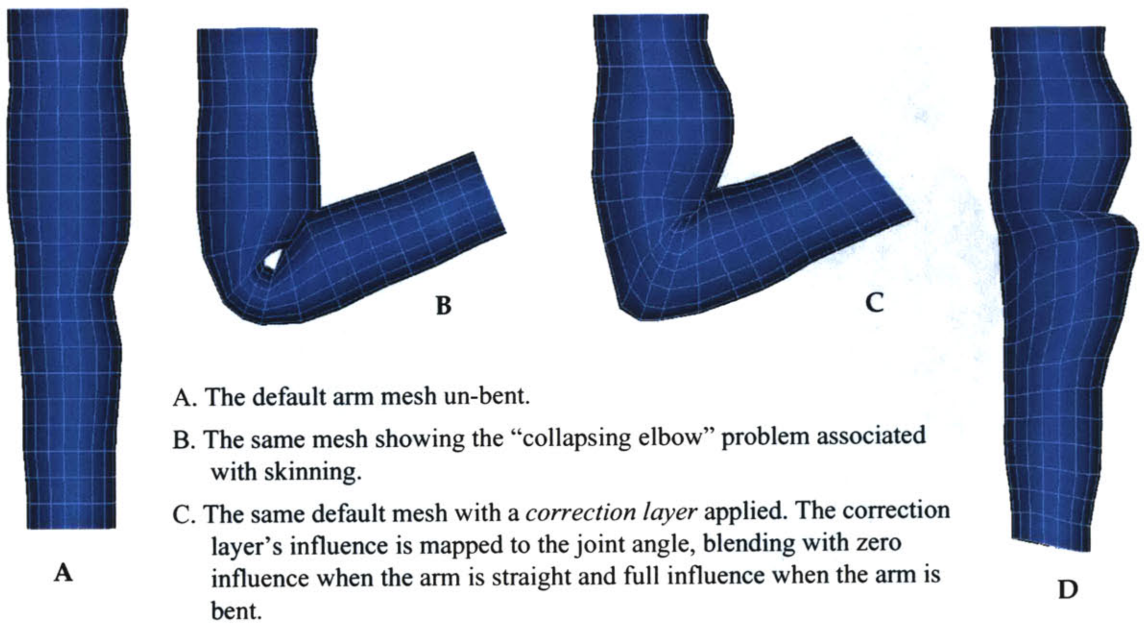


Fig 3-13 The same layers applied to an old face

Another interesting application of layering is to correct the shortcomings of the general skinning algorithm [13][21][25]. Because of the way skin deformation is calculated in response to joint rotations, it fails to preserve volume in the mesh around the joint, resulting in the "collapsing elbow" problem. This effect becomes increasingly noticeable at large joint angles and is illustrated in Fig 3-14. A second shortcoming of the skinning formulation is that, short of inserting special "muscle" bones into the skeleton, it doesn't allow for the muscle bulge you would expect when a joint is articulated. MorphLayers can be used to compensate for both of these shortcomings.

With MorphLayers it is possible to locally correct the breakdown of the skinning algorithm. By modulating the blend of the MorphLayers based on the joint angle, appropriate corrections can be made to the base mesh where it collapses. This application is inspired by the recent work of Lewis, et al [21]. To correct for the skinning breakdown in the original mesh (B in Fig 3-14), a "proper" deformation is sculpted in the mesh at the extreme of the articulation (C), the joint is then straightened, and the resulting mesh (D) is used as a MorphLayer that blends based on the joint angle.



A. The default arm mesh un-bent.

B. The same mesh showing the "collapsing elbow" problem associated with skinning.

C. The same default mesh with a *correction layer* applied. The correction layer's influence is mapped to the joint angle, blending with zero influence when the arm is straight and full influence when the arm is bent.

D. The *correction layer* that is blended with A to give the results in C.

Fig 3-14 The application of MorphLayers to minimize elbow collapse and mimic muscle expansion

It would be possible to extend this idea into a multidimensional MorphSpace to correct for problems that arise in multi-axis articulation of a joint. The deformation at the elbow, for example, needs to deal with both the blending of the elbow and the pronation/supination of the forearm. These two angles can be mapped to a two dimensional "layered morph space" to correct for the skinning breakdown during these articulations. This would require the artist to design three *correction meshes,* one for a fully bent elbow, one for a fully supinated forearm, and one for a fully bent/supinated arm.

3.3.6 Updating the Morphology System

The Morphology System is an *updateable* system within the creature. This means that at every tick the system get an update call from the creature telling it to update its internal systems. When the Morphology System gets this call, it updates the installed component systems in this order: MorphSpace, MorphAnimations, and then the MorphLayers. The ordering of the update is important because of the hierarchical nature of the Morphology System. At the top level, the MorphSpace determines the base characteristics of the body. At the next level, a MorphAnimation may be activated telling the body to transition to a certain node temporarily, possibly for expressive effect. Then on top of these changes to the body, one or more layers can be added that reflect emotion or other state. Once all these blends have been made the body is ready to be output to the screen by the graphics system.

The important question to ask about updating in the Morphology System is "when should it be done?" The question is important because, with a graphically complex character, the morphing calculations quickly become the limiting factor in the entire creature update. The updating scheme determines the trade-off between consistency of framerate, and the smoothness of the morphing.

As a rule of thumb, the frequency of the blending (every tick versus every fifth tick, for example) should depend on the speed of the morphological change the character is undergoing. The slower and more gradual the changes the less frequently the MorphSpace needs to re-blended in order to maintain a smooth development. For slow morphological changes like aging, it does not make sense to update the MorphSpace at every clock cycle. A better scheme that would be to update every fifth, tenth, or nth frame. The MorphSpace allows the user to specify this update frequency. Table 1 shows the results of different updating schemes on overall frame rate.

The gradual nature of MorphSpace transitions allow significant latitude when updating. MorphAnimations, however, have more pressing morphing needs. If an animation is specified to complete in 60 frames and must be synchronize with a motor animation, it is hardly acceptable for the MorphAnimation to blend every 20th frame. This would show a total of three in-between frames during the transition, hardly a smooth blend. Specifying the blend frequency in a MorphAnimation is really specifying how many frames of the transition are allowed to be dropped. Testing has show the maximum tolerance for dropped frames is about 10% of the number of frames in the animation (subjectively measured by observing the relative "smoothness" of the morphing animation). This updating scheme, in practice, produces minimal computational savings but more economical schemes can be used as needed.

3.4 Summary

This chapter details the inner working of the Morphology System and its essential tie ins with the Synthetic Characters C4 architecture. The combination of MorphSpaces, MorphAnimations, and MorphLayers gives the character designer the ability to design a wide range of morphologically pliable creatures. This design process, however, is not without considerations. The next chapter investigates these considerations and develops a workflow for designing morphable characters.

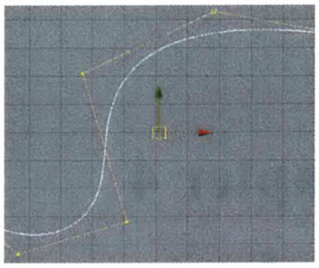
Chapter 4
Designing for Morphology

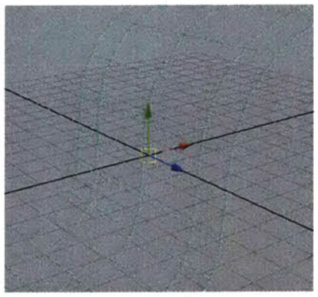
Designing for character morphology is not an haphazard undertaking. The nature of the application places strict requirements on the designer or artist. Two characters cannot be modeled independently and expected to morph with each other. Each character must be developed with morphology in mind from the outset. This chapter summarizes the advantages and disadvantages of common modeling techniques and outlines a workflow for successfully designing morphable characters.

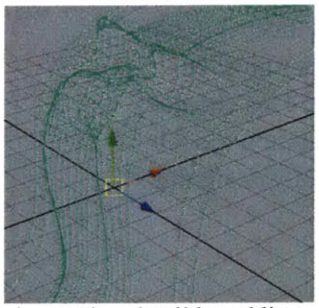
4.1 Modeling

The most important consideration when designing characters for morphing applications is maintaining homogeneous topology across morph targets. Designing with this consideration in mind eliminates the onerous problem of vertex correspondence described in Section 1.3. Even high-end 3D modeling packages that implement multi-target morphing impose this requirement of consistent topology on users [2][3].

With the ever-growing selection of modeling tools available to the designer, the question becomes what is the best way to produce models for variable morphology? We will look briefly at the features and limitations of these three primary modeling techniques: non-uniform rational B-spline surfaces (NURBS), polygon modeling, and subdivision surfaces. After looking at the advantages and limitations of these techniques, we will establish a workflow for easily developing complex models that work around the vertex correspondence problem. It is important to be clear that no matter what method is used to model, the ultimate output needs to be a polygon mesh of consistent vertex count. This means that parametric surfaces like those produced with NURBS need to be tessellated into polygon meshes before exporting for use in real-time.


a simple NURB curve


a NURB surface


increasing detail by adding isoparams

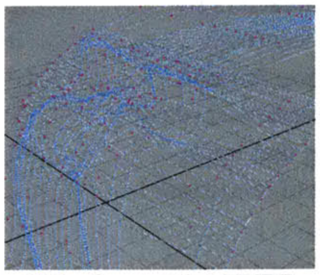

CVs of a complex NURB surface

Fig 4-1 NURBS fundamentals

4.1.1 NURBS

NURBS are characterized by their ability to model complex organic surfaces. This organic look is provided by the mathematical representation that underlies the surface, the B-spline. A non-uniform (knots can be irregularly spaced), rational B-spline is a curve, most often of degree three, which is controlled by the positioning of its control vertices (CVs) [27]. By adjusting the CVs, the curvature of the spline is modulated, but will always remains smooth. Extending the concept of the B-spline into three dimensions gives a NURB surface. These surfaces are controlled by the underlying curvature of intersecting B-splines, which are called isoparams [2][3]. Details are added to a NURB surface by increasing the number of isoparams that characterize the surface and by adjusting the CVs that shape these isoparams (Fig 4-1). By incrementally refining a surface with more isoparams a extremely detailed model can be constructed.

This method is extremely powerful and has been a standard technique in production modeling for years [2], but it has a number of limitations that make is less than suitable for designing morphable characters. When designing for variable morphology, the base model is generally the first model that is constructed. Subsequent morph targets are made by modifying this base model. When modifying the detailed base model, it is desirable to have high level controls points that allow for coarse adjustment to the overall shape of the model, while at the same time retaining the ability to go back in and modify details down to the finest level. A detailed NURB surface, like the one shown in Fig 4-2, offers only control over the details, there are no "coarse" control vertices that would allow the designer to modify gross shape attributes and then work his way into the details. Thus, once the finished base model is created, the designer is trapped in the details and cannot easily make high level changes to the structure of the model.

What is the work around? There is no straightforward technique with

NURBS. An attempt could be made to start with two identical course base meshes and then refine both of these to the desired level of detail by inserting isoparams as needed. But, to maintain topological correspondence, these isoparams would have to be inserted in exactly the same order on both models, taking care to always insert them between the same neighbors. Even if care is taken doing this, the chance of user error is so high that when the surfaces are tessellated into polygon meshes, there is a high probability that the meshes won't morph correctly.

This lack of hierarchical control over the model is the primary limitation of NURBS for modeling morphable characters. The other limitation of NURBS is that complex irregular shapes can only be created by attaching separate NURB surfaces, resulting in undesirable seams in the model [10]. Thus, instead of having one continuous polygon mesh when the model is tessellated, there are many pieces that appear to be attached but actually have seams located all over. While these seams are usually smooth and hidden, if they are not "welded" together after the surface has been converted to polygons then tears can occur during animation. In complex models, this welding can be extremely time-consuming. Additionally, there is no guarantee that the modeling software will order the welded vertices in the same order from one model to the next. Modeling in polygons from the beginning eliminates these problems.

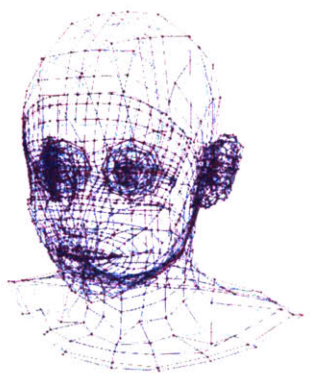


Fig 4-2 Detail of a complex NURBS model

4.1.2 Polygonal modeling

Polygonal modeling is a simpler method than modeling with NURBS. There is no tessellation stage to wrestle with and seams are no longer a concern; but in exchange, polygon modeling doesn't produce the smoothness or resolution found in NURBS models. In polygonal modeling, the designer starts with a polygon primitive such as a square and then uses operations such as extruding and splitting faces in conjunction with pushing and pulling the vertices to increase the detail. What results is a continuous polygon mesh with no seams and deliberately placed details. This method produces models that are ideal for low-resolution characters, but is tedious and time consuming when working in higher resolutions. Modifying a low resolution model into morph targets is relatively straightforward and just involves pushing and pulling vertices into their new shape. This becomes more complicated as the model increases in detail. Increasingly complex models again feel the need for high level control over the model. Polygon modeling is well suited to making seamless, low-resolution models but, by itself, is inadequate for making smooth detailed surfaces. Subdivision surfaces solve this problem.

4.1.3 Subdivision surfaces

Subdivision surfaces, specifically hierarchical subdivision surfaces based around the Catmull-Clark subdivision scheme [2][8][41], have proven to be the ideal technique for developing characters with variable morphology. A subdivision surface is a surface that results from applying a subdivision algorithm to a polygonal model, thereby increasing surface tesselation and smoothing polygonal edges [8][10]. In essence, it creates a surface that represents the maximum limit of this smoothing and use the original polygonal vertices as control points for the surface, very similar to the CVs in NURBS [8]. This smoothness gives them all the advantages of NURBS and, having a polygonal representation at their base, gives them the seamless qualities gained from simple polygon modeling. The most useful feature for variable morphology however, is the ability to hierarchically edit the surface at different levels of refinement. This gives the designer the ability to edit the surface at the coarsest level, that of the base polygon mesh, and then through increasing levels of refinement.

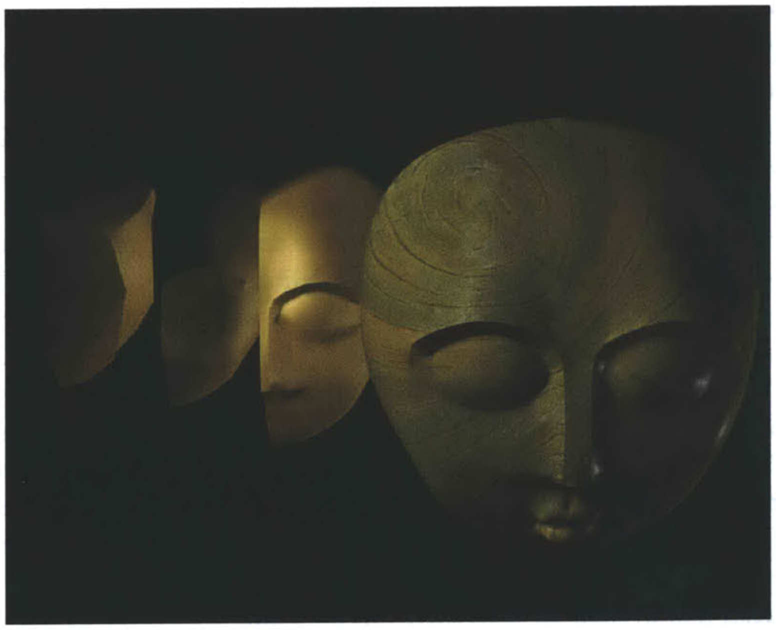


Fig 4-3 The use of subdivision surfaces in designing the Zulu Warrior

4.1.3.1 A general workflow with subdivision surfaces:

1. *Starting with a base polygonal mesh establish a general shape using as few vertices as possible. This mesh will serve as the control mesh for the subdivision surface.*

2. *For characters with symmetry, divide the mesh in half and make an instance of it. Scale this instance by -1 about the axis of symmetry (effectively mirroring the object).*

3. *Subdivide and add detail as necessary, try to finalize the details on this level before moving to next level of refinement.*

4. *Finalize the base character. This model will serve as the point of departure for subsequent morph targets.*

5. *Identify features that should maintain vertex correspond from one model to the next. For example, the eyes, nose, and mouth should be defined by the same vertices across all models. Models that do not maintain feature correspondence will not morph correctly.*

6. *Use vertex painting to mark these features.*

7. *Rework the model starting with the coarsest level of detail and progressively refining the mesh*

8. *Ensure the painted vertices are reconstructed into the appropriate features.*

9. *Finalize morph targets.*

This technique ensures a consistent vertex count as long as the topology of the base mesh is not modified when modelling the morph targets. The only limitation associated with this modeling technique is a sometimes less than optimal polygon count that results from subdividing the entire mesh. This can be seen when looking at the back of the Zulu Warrior' head in Fig 4-4. The back of the head is devoid of detail yet is as densely refined as the front of the face because the entire mesh was subdivided. This can be eliminated by carefully choosing the location of details and selectively refining the mesh only in those areas, in this case only the front of the face.

4.2 Skeletons and Animation

Constructing skeletons for morphable characters is difficult only because the joints of the target skeletons need to match the joints in the base skeleton exactly in both hierarchy and in naming. Any additions or changes to the hierarchy are not handled by the Morphology System. The position and rotation of each joint may be modified as needed to achieve the desired structure. In general, the best way to maintain consistent hierarchy and naming is to import and directly modify the base skeleton.

4.3 Optimizing Characters

The computational expense of morphing characters can quickly become the limiting factor in system updates. Because of this, characters must be carefully designed to maximize appearance and functionality while minimizing the expenses incurred in morphing. Below are techniques for optimizing characters for morphology.

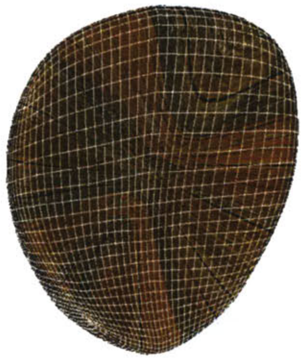


Fig 4-4 Over refined portion of the head

1. Optimizing polygon count
In general, the number of polygons that make up a character is directly proportional to the amount of time that it takes to display the character on the screen. The more polygons, the more resource that are required to move it on the screen, and the more resources that are required to morph the character. When designing characters, the amount of detail that the character should have needs to be propor-

tional to the its importance in the scene. If the character is never going to fill more than ten pixels on the screen, there is no point in modeling a complex 20,000 polygon body.

At the same time, even complex characters should be designed to optimize their polygon count. Fig 4-4 shows wasteful use of polygons. Far too many polygons are used to achieve the desired amount of detail. This, of course, is the product of globally subdividing a mesh and can be worked around using local subdivision. Small saving such as these, applied across the entire body, can reduce polygon counts by up to 40% without noticeably degrading the appearance of the character.

2. Minimizing Transform Controllers
Section 3.2 describes the anatomy of a virtual character and how Transform Controllers are used to represent skeletal joints and the position/orient of meshes. When many meshes are used in a character's body, the list of TransformControllers gets increasingly larger (one for each additional mesh). The longer the list, the longer the morphing times because every TransformController in the list is iterated through and blended. The list can easily become quite large when creating a detailed segmented character like the Zulu Warrior. Optimizing the TC list is possible if there are meshes in the body that are don't animate independently of a joint and are all linked to the same "bone". In the Zulu warrior, for example, there are 28 separate meshes for the dreadlocks which are all linked to the skull. None of the dreadlocks move independently and so there is no good reason to have 28 meshes when the exact same geometry could be represented with a single mesh. Simply collapsing all these meshes into a single mesh reduces the TC count by 27 (as well as the mesh count).

3. Optimizing morphLayers for skinned characters
Skinned polygons are expensive to display on the screen and doubly expensive to morph. Because the associated expense it so high, skinned characters need to be carefully designed. If MorphLayers are going to be used for something like facial expressions, it does not make sense to have the entire skin being passed in to each morphLayer. A better approach would to passing in a smaller "face" mesh. This can be accomplished by splitting the skin in two, the first the body and the second the face. One problem with this method is that a seam is created at the split. However, if the vertices in the seam have the exact same bone weights, they will always remain coincident and never tear apart. With a construction like this, the head mesh can be used as the only input to the MorphLayer, sparing the expense of needlessly iterating through and blending body vertices for every layer

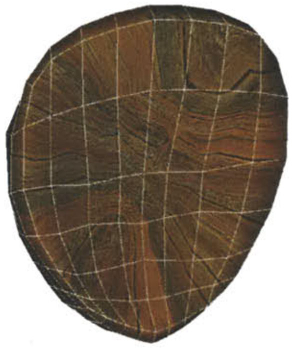


Fig 4-5 A more appropriate subdivision scheme

Carefully designing characters for morphology is an involved process with many restrictions. Using the procedures outlined in this chapter, it is quite easy to create character that morph reliably and are computationally efficient.

Chapter 5
Evaluation and Future Work

5.1 Evaluation

The work in this thesis is best evaluated by its ability to meet the requirements of a general character morphology system as outlined in Section 1.1. To restate these goals, a morphology system should first give the creature control over developmental changes associated with its body. These developmental changes are generally long-term, gradual changes that will exhibit themselves through prolonged interaction with the environment. These changes should also manifest themselves in a qualitative change in the creature's motion. Secondly, a morphology system should give the character designer additional expressive control over the appearance of a character. In this capacity a morphology system serves to augment the standard character animation tool set. This functionality might manifest itself in the ability to control localized deformations of a character's body for use in facial expressions or more imaginative applications. Thirdly, the system should be tightly integrated into a higher level behavior system. Finally, I would add that performance of the system is an important point worth evaluating. Starting with the first of these we will look to assess (albeit subjectively) the success of this Morphology System in accomplishing these goals.

5.1.1 Development

The MorphSpaces as described in section 3.3.3, handle all issues of development. The integration of the MorphSpaces with adverbed motor animations produces remarkably compelling results. The blending of sample animations does an excellent job at approximating the changes in a character's motion throughout development.

The primarily drawback associated with this MorphSpace implementation is the additional workload that it places on the animator and modeler. For every dimension in the MorphSpace, 2^n animations are required, or for locomotion (because bearing is produced by blending three animations), $3*2^n$ animation are needed. For the simple BlueMan demonstration (see Appendix A), 12 walk-cycle animations were needed, plus 12 stand-to-walk transitions, plus 12 walk-to-stand transitions, etc. It can easily be seen that this implementation does not scale well beyond two dimensional MorphSpaces. Of course, if skeletons do not change proportion across nodes, then one set of animations could be used for both nodes used as long as uniqueness of motion is not important. Alternatively, if functional animations are all that are required, the work of Gleicher [15] would effectively retarget a base set of animations across many different skeletons.

5.1.2 Expression

MorphLayers and MorphAnimations are the primary vehicles for additional expressive control of a character. The success of the Morph System's architecture is that it allows these two component to change independently of the MorphSpaces. In well designed characters with feature correspondence, the designer has tremendous control over the creature's expression. When using LayerMorphNodes, the expressive ability of the system is enhanced by its ability to extrapolate changes beyond the original intent of the designer, often resulting in interesting, caricature-like effects.

The main limitations of the MorphAnimations and the MorphLayers are the additional computational cost and the associated effect on the consistency of the overall framerate. It is difficult to maintain a consistent framerate when updating these components only when they change, but it is often too expensive to update them all the time. The resulting slowdown is often not noticeable in MorphLayers where the changes can be more gradual and spread across many frames. MorphAnimations, however, demand a frame by frame update. Running performance tests on a creature that has both a Morphspace and a MorphAnimation installed shows a 10% decrease in framerate when the [layer]MorphAnimation is playing. While not drastic (30 Hz v. 34 Hz), this can result in an noticeable framerate hiccup. To stabilize the framerate, one possible solution would be to run a "noop" MorphAnimation in the background that blends values but doesn't output to the screen. This of course wastes computation but if the framerate slowdown is too noticeable may prove an adequate alternative.

5.1.3 Integration

The Morphology System has been fully integrated into the C4 behavior system maintaining the appropriate abstraction layers in the process. Because the Morphology System was designed to communicate with the rest of the behavior system only through the internal blackboard, functional changes to the C4 architecture are largely unseen by the Morphology System. This abstraction allows the behavior architecture to be continually refined without having to modify the internal workings of the Morphology System at every change. The value of this modularity was seen recently when the behavior system received a substantial upgrade to C4.1. In this upgrade many changes were made to the inner workings of the Action System. Despite these sweeping changes, the Morphology System required only small modifications to ensure compatibility.

5.1.4 Performance

The following tables summarize the performance of the Morphology System under various test conditions.

Table 1: MorphSpace updating schemes

update frequency	*framerate (Hz)*	*% change*
always	28	( )
every 2nd cycle	33	+15
every 3rd cycle	35	+20
every 5th cycle	37	+24
every 10th cycle	39	+28
every 20th cycle	40	+30
never	40	+30

discussion: This test was conducted on the Zulu Warrior, the most complex character in this thesis, to test the worst case performance of the MorphSpaces. The results show a significant savings when using any update scheme other than "every cycle". When choosing an update frequency for a MorphSpace, the frequency and magnitude of blending changes from the behavior system need to be considered. Ultimately, the updating scheme should be decided on a case by case basis.

Table 2: MorphSpace Performance with Different Blending Specifications

information to blend	*framerate (Hz)*	*% change*
baseline: meshes, materials, skeleton, boneweights	82	( )
meshes, materials, skeleton	89	+7.8
meshes, skeleton	90	+8.8
only meshes	96	+14.5
only materials	101	+18.8
nothing	103	+20.3

discussion: This table summarizes the performance cost associated with blending different aspects of a character's morphology. The tests were run on the BlueMan of Fig 3-1. His body consists of one 800 polygon skinned mesh, two linked eyes, and 17 bones. The difference between the first and second entries in the table shows the cost associated with blending the bone weights, which is quite significant. Test have shown however that when blending between morphologically similar character, one

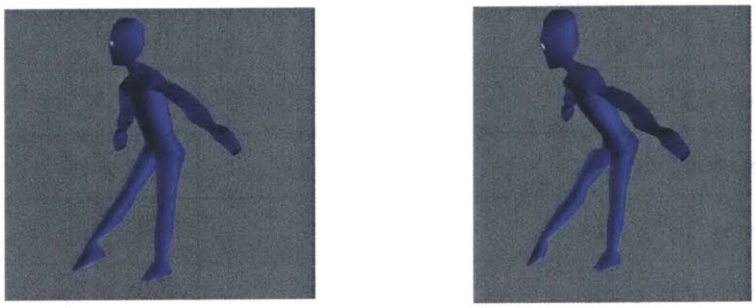


Fig 5-1 Blending boneweights vs. not blending boneweights

set of bone weights often provides a good approximation across all bodies.

Table 3: Performance with Multiple MorphLayers

configuration	*framerate (Hz)*	*% change from baseline*
baseline - no layers	33.8	( )
1 layer	31.7	-6.2
2 layers	29.3	-12.1
3 layers	27.7	-18.0

discussion: This table shows the system performance with a MorphSpace installed and increasing numbers of MorphLayers. The tests were conducted on the Zulu Warrior's head using the expression layers from Fig 3-12. For testing purposes both the MorphLayers are the MorphSpaces are updating every cycle. These results show a nearly constant computational cost associated with adding additional layers.

Table 4: Layer Optimization

	framerate (Hz)	*% change*
multiplying all vertices	34.26	( )
multiplying only nonzero values	34.56	+0.87

discussion: This small optimization proves to be just that, small. The creation of LayerNodes produces many vertex entries that are zero. This test was run on a LayerNode that contained approximately 25% zeros. This optimization would prove slightly more significant with a more sparsely populated LayerNode (~1%).

5.2 Future Work

There are additional feature that could be developed to extend the functionality of the Morphology System. In it current incarnation, the Morphology System addresses texturing issues at only a superficial level. Essentially, the textures a character starts with at runtime are the textures it keeps for the duration of its life cycle. Currently, the system allows the user to modify the material properties that underlie the textures, such as diffuse color, specularity, and opacity, but does not address the issue of blending texture maps or UV coordinates. The latter is a trivial implementation, but the former requires an investigation into both the low-level engineering issues associate with morphing textures and the texture design workflow. This addition would help to reenforce the believability of a character's development cycle.

Beyond texturing, an intelligent scheme for optimizing the blending of LayerNodes could be developed. In the current implementation, each mesh in a LayerNode contains many vertices with zero values that have no effect on the overall blend, but we are forced to query each vertex to see if it has non-zero information. This query becomes needlessly expensive in meshes with high vertex counts. An intelligent solution should localize only the vertices that change and blend these values, spending no computation on parts of the mesh that don't change.

Finally, a system for sharing MorphNodes across creatures needs to be developed. It certain instances creatures will share the same base geometry and use similar nodes in their Morphology Systems. Currently, each creature loads and maintains its own individual copy of these nodes. This is necessary for unique, creature-specific nodes, but can become quite memory intensive when multiple creatures begin loading redundant copies of the same geometry file. In the AlphaWolf project currently under development, five creatures will be running Morphology Systems using similar base geometry and target nodes. A pathway for referencing common nodes would significantly reduce the Morphology System's memory consumption.

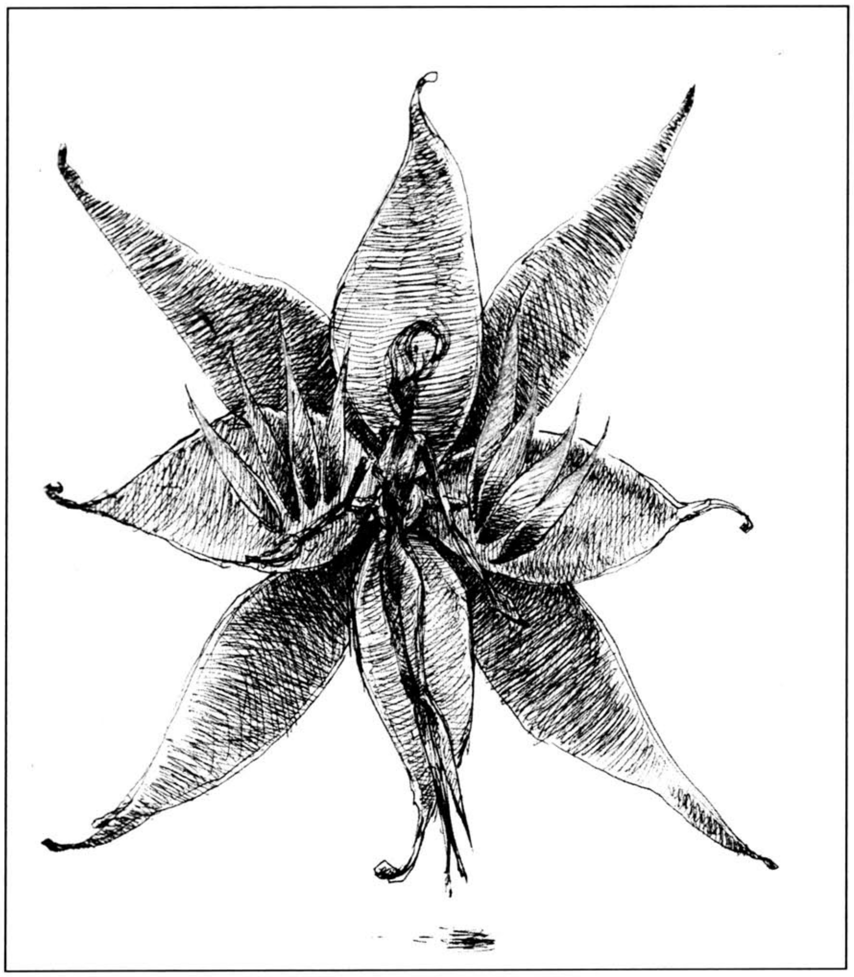
5.3 Conclusion

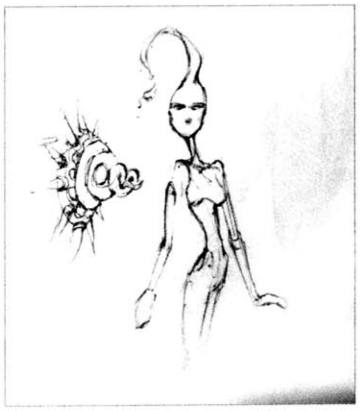
This thesis presents the details of the Morphology System that I developed for inclusion into the Synthetic Character's C4 behavior architecture. The Morphology System seamlessly interfaces body changes with a character's behavior and motor systems. This integration gives a character the ability to respond appropriately to it environment in every way. The intelligence and learning in the behavior system allows the character to make rational decision based on its knowledge of the world. Over prolonged interactions, these decisions change the internal state of the character, which, in turn, manifests changes in the character's morphological appearance. These changes in the Morphology System affects how the character moves and appears. Through these small steps we have moved closer to intelligent creatures that can live and grow in a computational environment.

Appendix A
Character Sketches

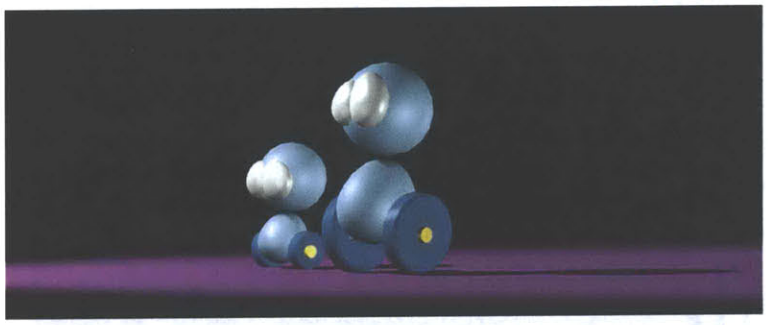
lady | flower

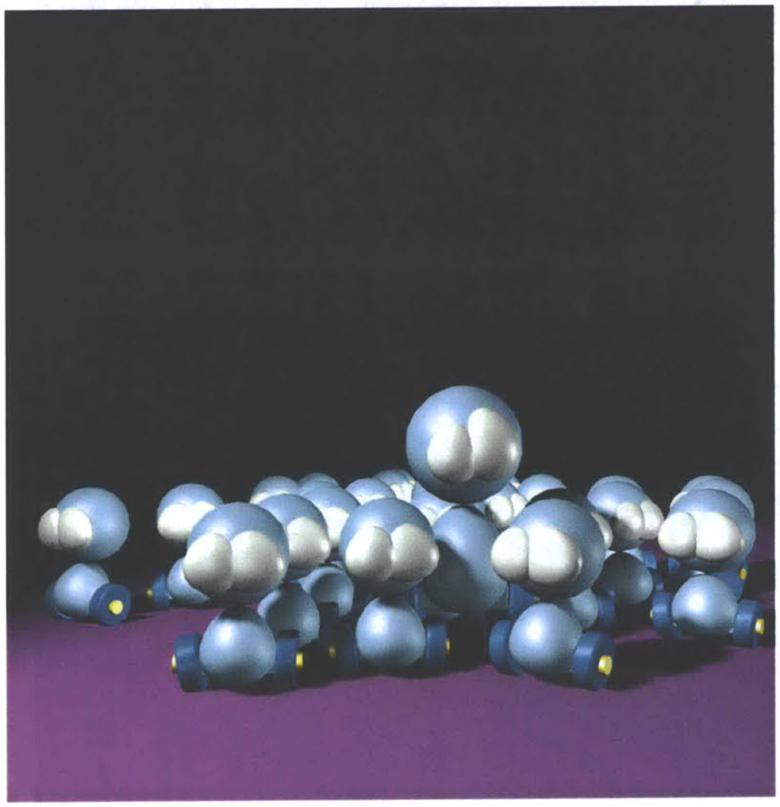
The lady was the first creature that I employed during the development of this thesis. I made her acquaintance in Dublin, Ireland, at the National College of Art and Design in November 2000. In exchange for passage across the Atlantic, she agreed to five years of indentured servitude. While in my employment she posed for many drawings and computer models. I also used her for early behavior system tests. Her usefulness to me expired midway through the development of the Morphology System, so I lobotomized her. She now lives in a old peanut butter jar in my kitchen.





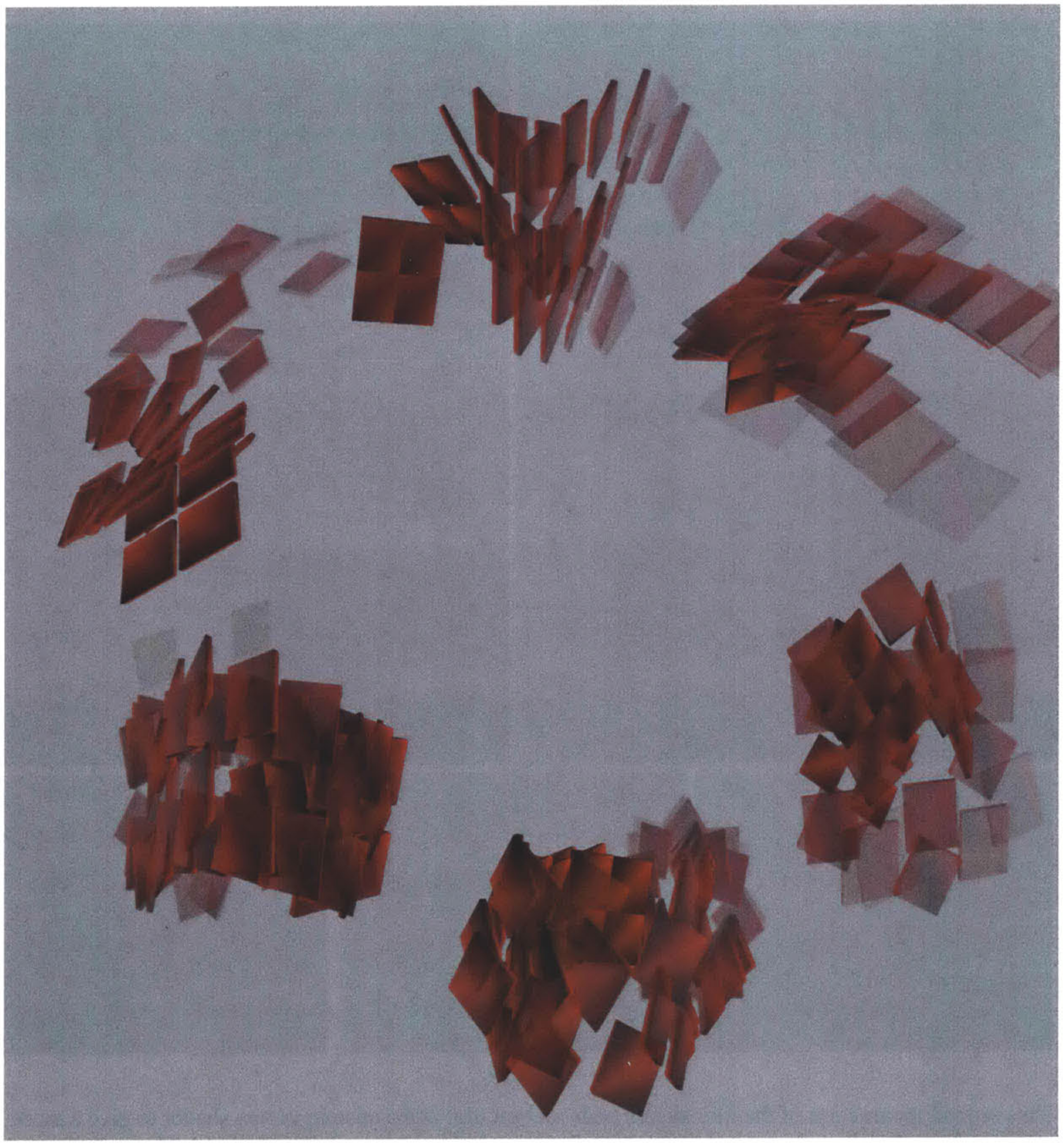
These images show the lady and her flower in an environment that is morphing randomly as she moves through it. The last image shows a screenshot of her flower morphing as it undulates.





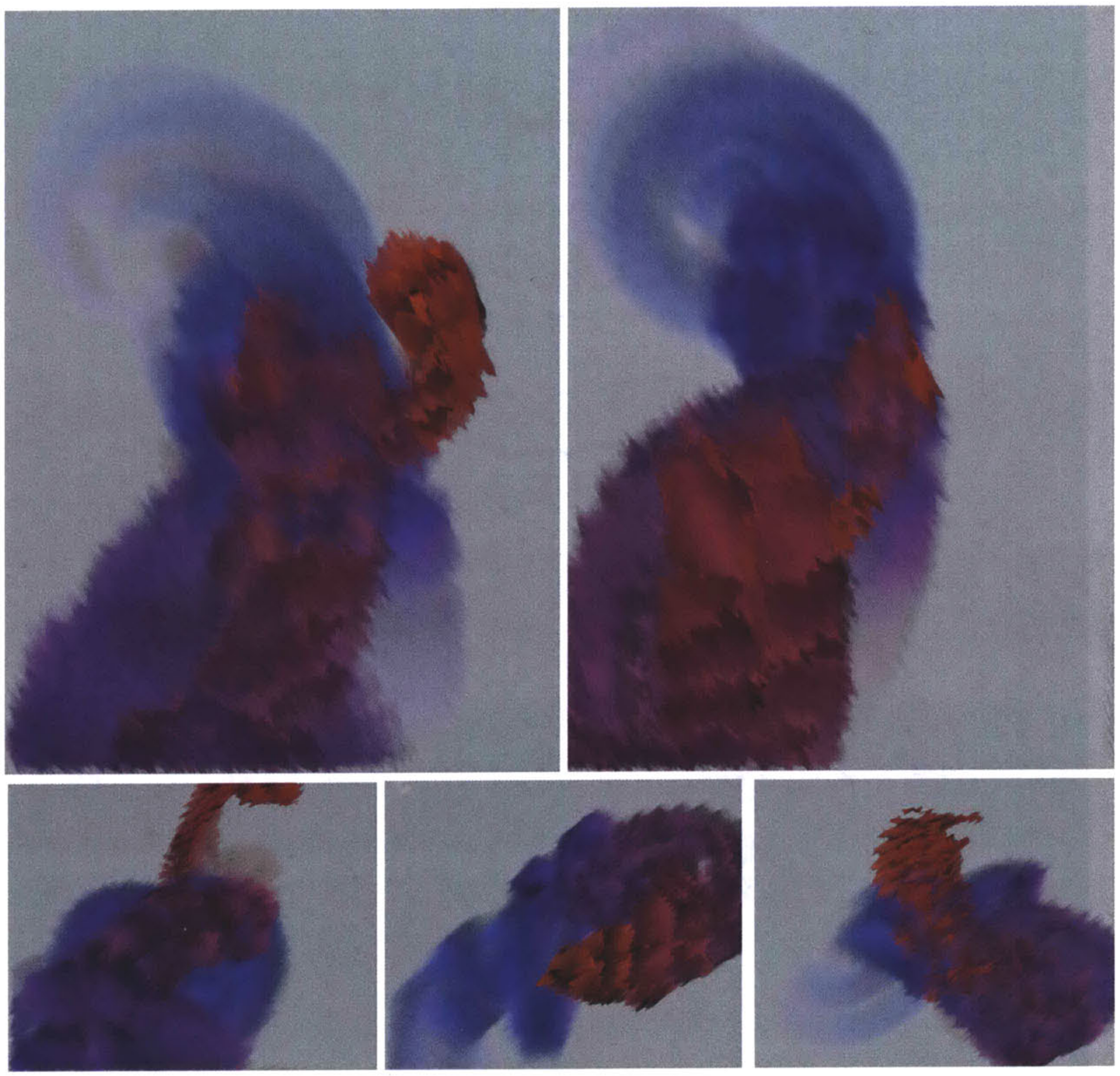
CarcaroclesChubitensis

After the lobotomy, the lady would just sit in her peanut butter jar all day so I needed new creatures to test the Morphology System. One day I was jogging along the Minuteman trail and saw two small creatures scurry across the path. After rooting around in the underbrush I was able to ferret both of them out. Fortune smiled on me, I now had in my possession two strange blue creatures for my tests. I showed them around at the lab and nobody had ever seen anything like them (all the girls thought they were cute though). I did a Google search on the internet and found out they were Carcarocles Chubitensis, the prehistoric ancestor of the Great White Shark. I stored them in a shoe box at home and brought them into the lab for testing. They were used primarily to test the skeleton blending, making sure than animations would still playout smoothly as the skeleton morphed. Because their skeletons are made of cartilage, they were quite resilient and could tolerate far more testing than most mammals. They also helped me with the development of the MorphAnimations and the MorphSpaces. One morning as I was getting ready to feed them breakfast (many trips to the harbor to get chum off the fishing boats), I opened the shoe box and found 20, maybe, 30 mini-chubs running around. I sold a few to friends and gave the rest to the Humane Society.



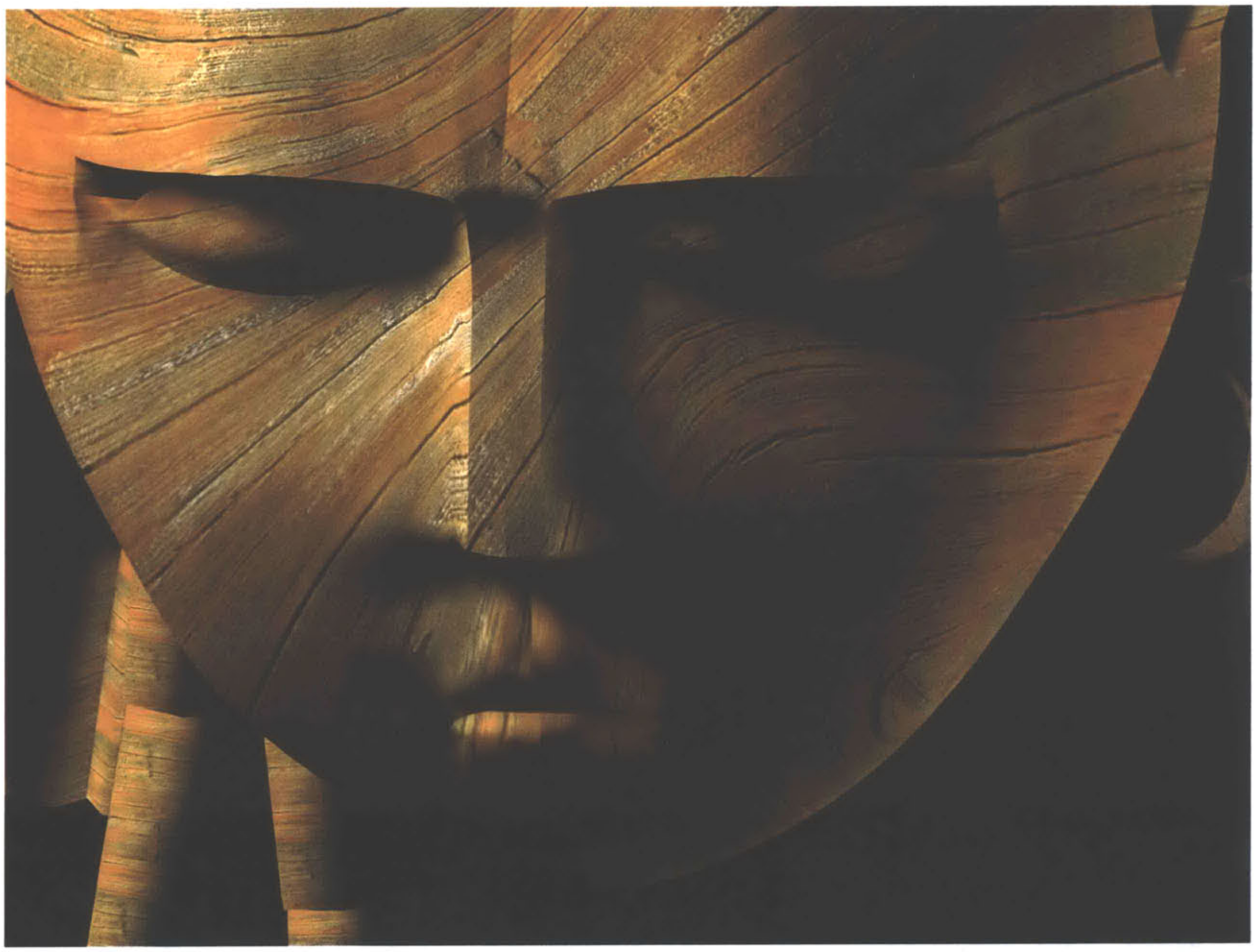
Tiles

This small sketch was inspired by the motion the flower uses to follow the lady. I removed the flower and the lady and put 30 simple placeholders in their stead. The motion of these 30 tiles following each other was quite compelling. I refined the sketch by adding a morphology system to each tile and blending a number of shape transitions between the first tile and the last tile.



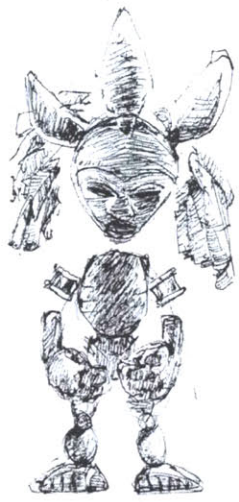
Tiles II

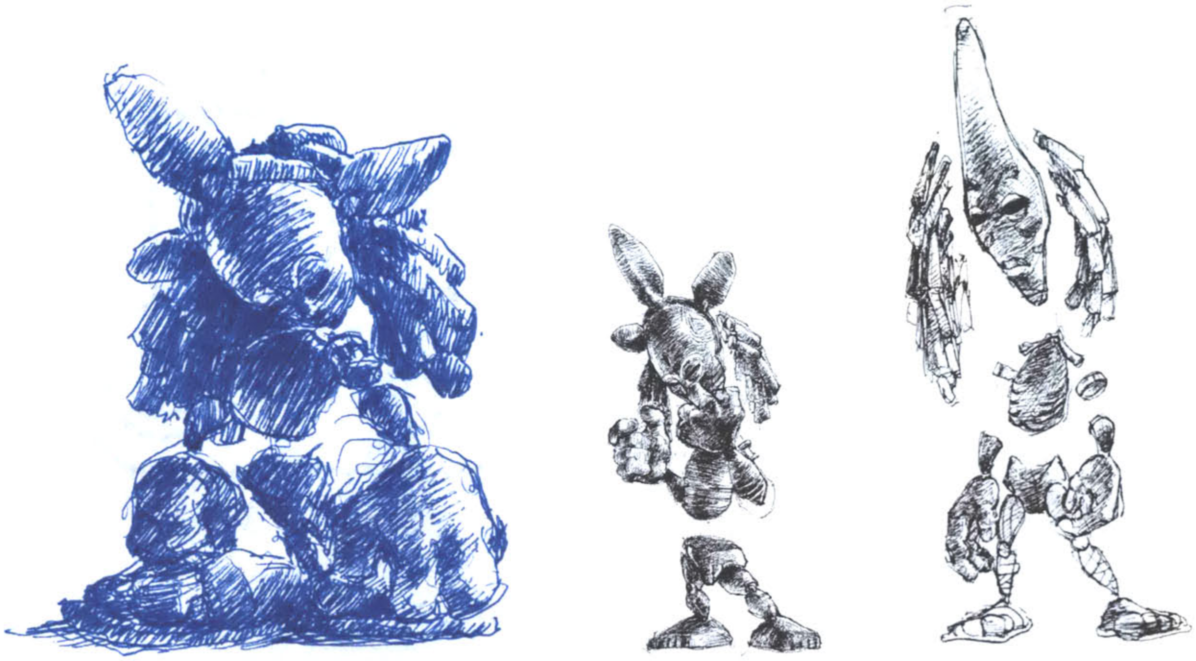
This second incarnation of the tiles sketch adds motion blur and a custom vertex shader to give a more ethereal feel. The tiles are created in the same way and have similar motion, but are able to play out a series of MorphAnimations with each tile morphing in succession. The result is a cascading propagation down the chain of tiles.

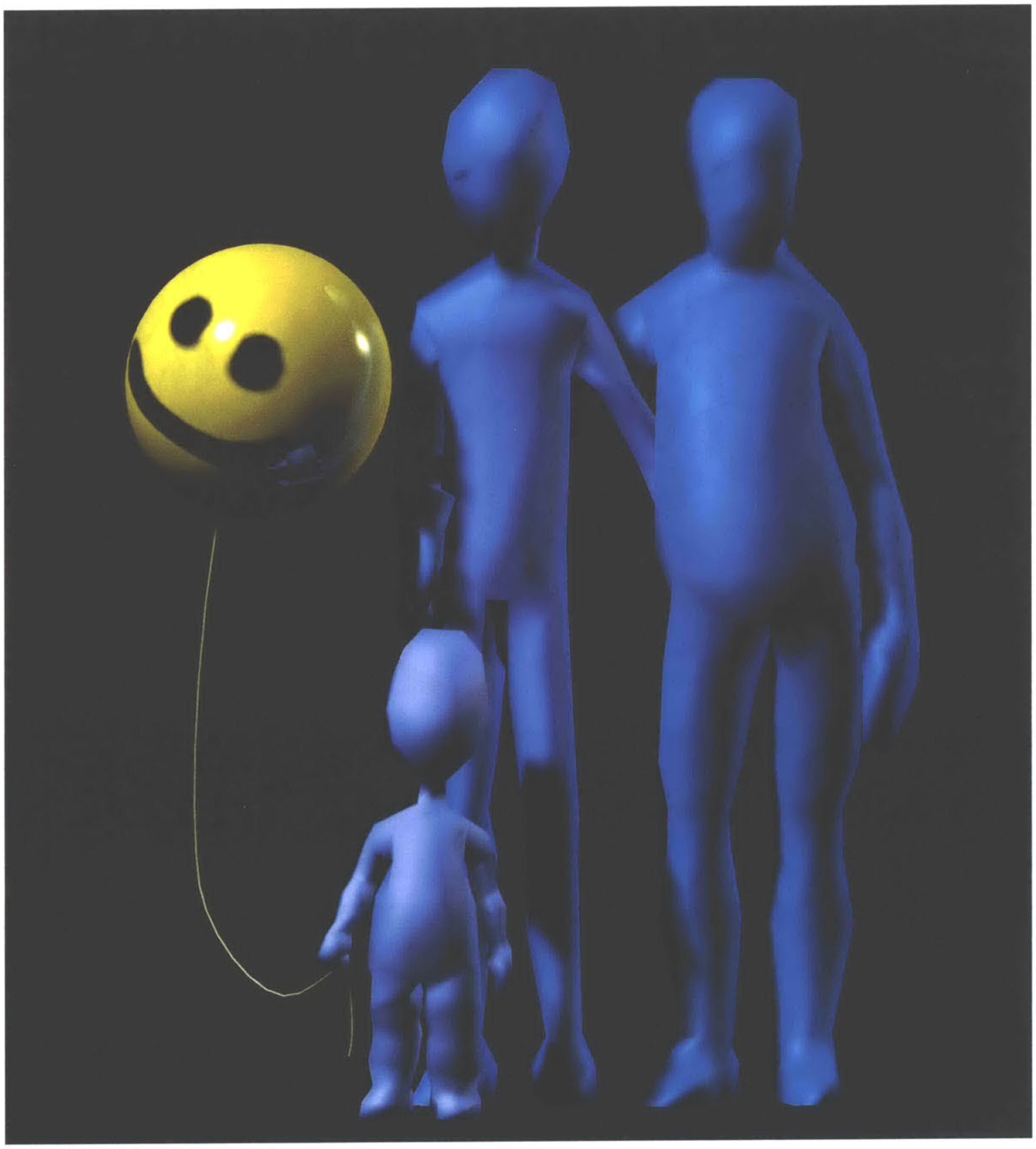


Zulu Warrior

The Zulu Warrior is the final character created for this thesis. He was inspired by African tribal masks and carvings. A few of the early concept sketches on shown on the following pages....







FIN

References

[1] American Heritage Dictionary, Dell Publishing, New York, NY. 1994.

[2] Alias|Wavefront, *Maya V3.0,* 2000. For information: www.aliaswavefront.com.

[3] Autodesk Inc., *3DS Max V.4*, 2000. For information: www2.discreet.com.

[4] Boa, H. and Q. Peng, Interactive 3D Morphing. *Computer Graphics Forum* Vol. 17 no. 3. Boston, MA. Blackwell Publishers. 1998.

[5] Blanz,V. and T. Vetter, A Morphable Model for the Synthesis of 3D Faces. *In Proceedings of SIGGRAPH 99.* New York, NY. ACM SIGGRAPH. 1999.

[6] Blumberg, B. *Old Tricks, New Dogs: Ethology and Interactive Creatures.* Ph.D. Dissertation, MIT. 1996.

[7] Blumberg, B. and T. Galyeon,. Multi-Level Direction of Autonomous Creatures for Real Time Virtual Environments. *In proceedings of SIGGRAPH 95.* New York, NY. ACM SIGGRAPH. 1995

[8] Catmull, E., and J. Clark, Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design,* 10(6):350-355, 1978.

[9] DeCarlo D. and J. Gallier, Topological Evolution of Surfaces. *In Proceedings of Graphics Interface '96.* Morgan Kaufmann Publishers, New York, NY, 1996.

[10] DeRose, T., M. Kass, T. Truong, Subdivision Surfaces in Character Animation, *In Proceedings of SIGGRAPH 98*. New York, NY. ACM SIGGRAPH. 1998.

[11] Dennet, D. *The Intentional Stance.* MIT Press, Cambridge, MA, 1987.

[12] Downie, M. Behavior, Animation and Music: The Music and Movement of Synthetic Characters. Master's Thesis, MIT. 2000.

[13] Eberly, D., *3D Game Engine Design*, Morgan Kaufmann Publishers, New York, NY, 2001.

[14] Funge, J., X. Tu, and D. Terzopolous, Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. *In Proceedings of SIGGRAPH 99.* New York, NY. ACM SIGGRAPH. 1999.

[15] Gleicher, M. Retargeting Motion to New Characters. *In Proceedings of the 25th Annual Conference on Computer Graphics*. New York, NY. ACM Press. 1998.

[16] Hodgins, J and N. Pollard, Adapting Simulated Behaviors for New Characters, *In Proceedings of the 24th Annual ACM Conference on Computer Graphics* 1997.

[17] Isla, D., R. Burke, M. Downie, and B. Blumberg, A Layered Brain Architecture for Synthetic Characters, *IJCAI.* Seattle, WA, 2001. To appear.

[18] Itten, J. *The Elements of Color.* New York, NY. John Wiley & Sons, Inc. 1970.

[19] Johnson, M.P., Multi-Dimensional Quaternion Interpolation, In *ACM SIGGRAPH99 Conference Abstracts and Applications*, New York, NY, ACM SIGGRAPH. 1999.

[20] Kent, J., W. Carlson, and R. Parent, Shape Transformation for Polyhedral Objects. I*n Proceedings of SIGGRAPH 92.* New York, NY. ACM SIGGRAPH. 1992.

[21] Lewis, J.P., Cordner, M. and Fong, N. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. *In Proceeding of SIGGRAPH 00*, New York, NY. ACM SIGGRAPH. 2000.

[22] LionHead Studios. *Black and White.* www.lionhead.co.uk. 2000.

[23] Lui, L. and Wang, G. Three-Dimensional Shape Blending: Intrinsic Solutions to Spatial Interpolation Problems. *Computers and Graphics 23.* New York, NY. Pergamon Press. 1999.

[24] Maestri, G. *Digital Character Animation 2.* Vol. 1. Indianapolis, IN. New Rider. 1999.

[25] Magnenat-Thalmann, N., R. Laperriere, and D. Thalmann, Joint-Dependent Local Deformations for Hand Animation and Object Grasping, *In Proceeding of Graphics Interface '88*, Morgan Kaufmann Publishers, New York, NY,1988.

[26] Mendelowitz, E. *The Emergence Engine: A Behavior Based Agent Development Environment for Artists. In Proceedings of IAAI-2000.* Publisher not available. 2000.

[27] Mortenson, M. *Mathematics for Computer Graphics Applications.* New York, NY, Industrial Press, 1999.

[28] Perlin, K. and A. Goldberg, Improv: A System for Scripting Interactive Actors in Virtual Worlds. *Computer Graphics* Vol. 29 No. 3. 1996.

[29] Pina, A., E. Cerezo, and F. Seron, Computer Animation: From Avatars to Unrestricted Autonomous Actors. *Computers and Graphics 24.* New York, NY. Pergamon Press. 2000.

[30] Powell, M.J.D., Radial Basis Functions for Multivariable Interpolation: A Review. In *Algorithms for Approximation.* Oxford, UK, Oxford University Press, 1987.

[31] Pryor, K., *Don't Shoot The Dog: The New Art of Teaching and Training*, New York, NY, Bantam Doubled, 1999.

[32] Rose, C., *Verbs and Adverbs, Multidimensional Motion Interpolation Using Radial Basis Functions*, Ph.D. Dissertation, Princeton University, 1999.

[33] Rose, C., B. Guenter, B. Bodenheimer, and M. Cohen, Efficient Generation of Motion Transitions Using Spacetime Constraints, *In: Proceedings of the 23rd Annual Conference on Computer Graphics* 1996.

[34] Rose, C., M. Cohen, and B. Bodenheimer, Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Computer Graphics and Applications,* 18(5), New York, NY. 1998.

[35] Sims, K. Evolving Virtual Creatures. *In Proceeding of the 22nd Annual ACM Conference on Computer Graphics.* New York, NY. ACM Press 1995.

[36] Sun, Y.M., W. Wang, and F. Chin, Interpolating Polyhedral Models Using Intrinsic Shape Parameters. *The Journal of Visualization and Computer Animation,* Vol. 8. New York, NY. John Wiley & Sons, Ltd. 1997.

[37] Thomas, F. and O. Johnson, *The Illusion of Life: Disney Animation.* New York, NY. Hyperion. 1981.

[38] Tomlinson, B., M. Downie, B. Blumberg, and others. AlphaWolf. *In Abstracts and Applications, SIGGRAPH '01.* To appear.

[39] Tu, X. and D. Terzopolous, Artificial Fishes: Physics, Locomotion, Perception, Behavior. *In Proceedings of the 21st Annual ACM Conference on Computer Graphics*, New York, NY. ACM Press 1994.

[40] Turk, G. and J. O'Brien, Shape Transformation Using Implicit Functions. *In Proceedings of SIGGRAPH 99.* New York, NY. ACM SIGGRAPH. 1999.

[41] Zorin, D., *Subdivision and multiresolution surface representation.* Ph.D. Dissertation, CalTech, 1997.

123