3D Computer Animation Workshop

Siggraph'98 Course #34 Notes

Course Presenter

Michael O'Rourke
Pratt Institute

Administrative Assistant

Helen Koo, Pratt Institute

Animation Assistants

Abbey Klotz Pratt Institute
Helen Koo, Pratt Institute
Yukito Kurita, Pratt Institute
Khalida Lockheed, Pratt Institute
Gevel Marrero, Pratt Institute
William Sayer, Pratt Institute
Lina Yamaguchi, Pratt Institute
Eunmi Yang, Pratt Institute

Special Note to all Course participants:

Please bring your copy of these Course Notes with you to the Workshop; you will need to follow the Exercise portions of these Notes as you work at the machines. The Course Notes will also be available on line at each workstation.

Acknowledgements

I wish to thank the Siggraph organization and in particular, Harry Smith, Courses Chair, and Garry Paxinos, Creative Applications Lab Chair, for their help in organizing this Workshop. I also want to express my gratitude to all the people at Softimage for generously contributing their support to the Workshop. Finally, I want to thank all of my colleagues at Pratt Institute for their support, especially Rick Barry, Chair of the CGIM department, Helen Koo for her help in the preparation of these Course Notes and other Workshop materials, and all of my students and former students who helped as Animation Assistants to this Workshop.

Contents

Author Biography

Michael O'Rourke is an artist and animator and Associate Professor in the Department of Computer Graphics and Interactive Media at Pratt Institute, in Brooklyn, NY. His professional training in the arts includes an M.F.A. degree from the University of Pennsylvania. Following his studies, he was a Senior Research Staff Artist at the New York Institute of Technology Computer Graphics Laboratory, where he worked on personal animations and artwork as well as commercial animations, contributing to a Clio-award winning animation and a first-prize-winner at the Los Angeles Animation Celebration.

At Pratt, he is the senior faculty member and lead instructor in the animation program of Pratt's Department of Computer Graphics and Interactive Media. He is also the author of *Principles of Three-Dimensional Computer Animation* (W.W. Norton, 1998, Revised Edition). This book has also been printed in a Japanese language edition by Toppan Co., Tokyo, Japan. In addition to teaching, he actively pursues his own artwork, concentrating most recently on several series of prints and drawings. He has exhibited his work internationally, and was one of the featured artists in the Siggraph '97 *Ongoings* exhibition. He has also done several series of computer-aided sculpture and graphic works for the artist Frank Stella.

In addition to his experience as an artist, he has broad experience as an educator. His studies in this area were at Harvard University, where he earned an Ed.M. degree. In addition to his teaching at Pratt, he has taught Kindergarten, English as a foreign language in West Africa, and conversational French.

Contact Information:

Michael O'Rourke
Pratt Institute
Dept. of Computer Graphics & Interactive Media
200 Willoughby Ave.
Brooklyn, NY 11205

morourke@pratt.edu
http://pratt.edu/~morourke

Introduction

This course is a beginning level, hands-on workshop whose objective is to introduce participants to the principles and practice of high-end 3D computer animation. This is accomplished through a combination of lecture presentations and hands-on exercises using one of today's major high-end 3D software packages. Beginning with simple modeling, rendering and keyframing, participants progress through more complex techniques, including texture mapping, path animation, hierarchical animation, inverse kinematics and envelope surfaces. Participants learn both the underlying principles that are shared by all 3D computer animation software packages, as well as how these principles are implemented on one of today's important software packages.

Given the brief time available for the course, there is no expectation that participants will become proficient in the techniques presented. Instead, the hope is that they will come to understand the core principles presented and begin to see the possibilities of their implementation through the tutorial exercises. These exercises will remain on the workstations of the Creative Applications Laboratory throughout the week of the conference. Those with an interest in improving their understanding and skill beyond what is possible in the single day of this course can practice as much as they want during the week.

The course is divided into four parts, with each part consisting of a lecture presentation on the principles of 3D animation, a demonstration of how these principles are implemented on the software package used by the Workshop, and an extended exercise in which participants work with this software to develop a short animation utilizing those principles.

The written Notes for this course, printed here, follow the structure of the course itself – that is, four parts, with each part consisting of a brief lecture and an extended exercise. The exercises are in the form of command-by-command tutorials for *Softimage v3.7*, which is the software package used in the Workshop.

The text for the lecture portion of these Notes was adapted from the author's book, *Principles of Three-Dimensional Computer Animation* (W.W. Norton, 1998, Revised Edition. 288 pages, 332 illustrations. ISBN 0-393-73024-7. $55.00 USA). The illustrations for these Notes were taken directly from the same book. Please note that the illustrations retain the original numbering as they appear in that book, and are therefore not sequential as they appear here in these Notes. Readers interested in a more detailed explanation of the principles presented here, as well as many other principles of three-dimensional computer animation, may find it useful to refer to the book, *Principles of Three-Dimensional Computer Animation*, mentioned above, from which all this material is derived.

PART 1

Lecture 1

Coordinate Systems

Geometric Primitives

Transformations

Keyframing

Wireframe Preview

Parameter Curve Editing

Coordinate Systems

The most common way of locating points in three-dimensional space is a system of three **axes** placed perpendicularly to each other. The axes are referred to as the x axis, the y axis, and the z axis, as illustrated in Figure 1-29. Each axis has steps marked off along it at uniform distances allowing a point to be located in space by referring to the distances along each axis that one must move to get to that point. A movement in one direction is considered a positive movement, and a movement in the opposite direction is considered a negative movement. The numbers representing the distances are called the **coordinates** of the point. For example, a point might be located at (3, 2, -5). A special point in a coordinate system is the (0, 0, 0) point, which is thought of as the **origin** of the system. The entire system is called a coordinate system.

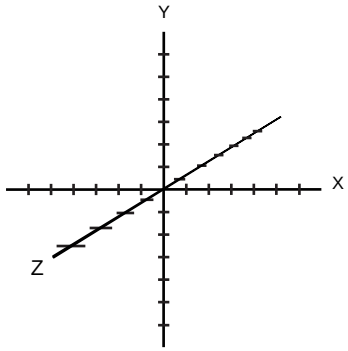


Figure 1-29. A three-dimensional Cartesian coordinate system.

The type of coordinate system we have just described is a **Cartesian coordinate system,** named after the French mathematician, René Descartes, who invented it. In three-dimensional computer graphics, all points and objects are located by specifying their coordinates in such a coordinate system.

Geometric Primitives

The simplest and most common kind of objects in computer graphics are objects such as the cube, cone, sphere, and cylinder. (Figure 1-36) Because they are very basic, they are called primitives, and because they have a strong geometric character, they are also called the **geometric primitives**. In addition to the primitives just mentioned, another common primitive is the torus, which is a donut-shaped object. In creating a specific geometric primitive, the user is usually asked to specify its location, its size, and sometimes other information. These are called the **parameters** of the object.

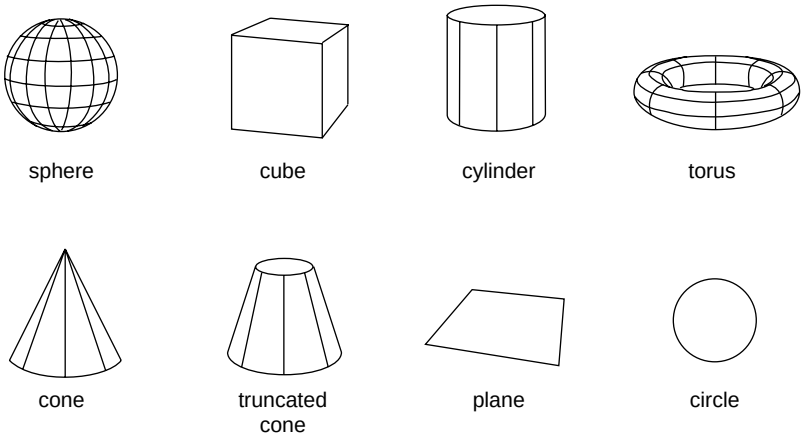


Figure 1-36. Some of the common geometric primitives.

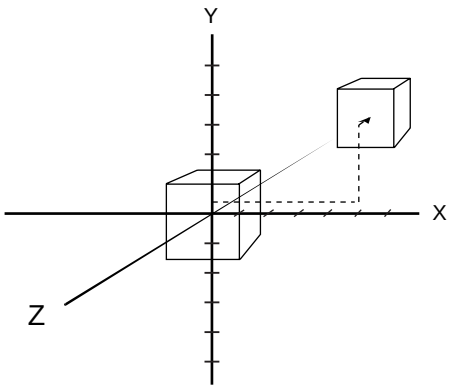


Figure 1-39. A simple translation of a cube.

Transformations

In positioning and manipulating an object in computer graphic space, there are three basic operations, known as the object's transformations. The first of these is the **move** transformation, also sometimes called a **translation**. (Figure 1-39) This consists in moving, or translating, an object some distance along each of the three axes either positively or negatively.

The second basic transformation is **rotation**. Like moves, rotations are thought of as taking place with respect to each of the three axes of the coordinate system, as if the axis were a skewer running through the center of the object. (Figure 1-40) Depending on which axis serves as the "skewer", we speak of an x rotation, a y rotation, or a z rotation. Rotating the object in one direction is a positive rotation and rotating in the opposite direction is a negative rotation. An object can be tilted into any position in three-dimensional space by some combination of x rotation, y rotation and z rotation.

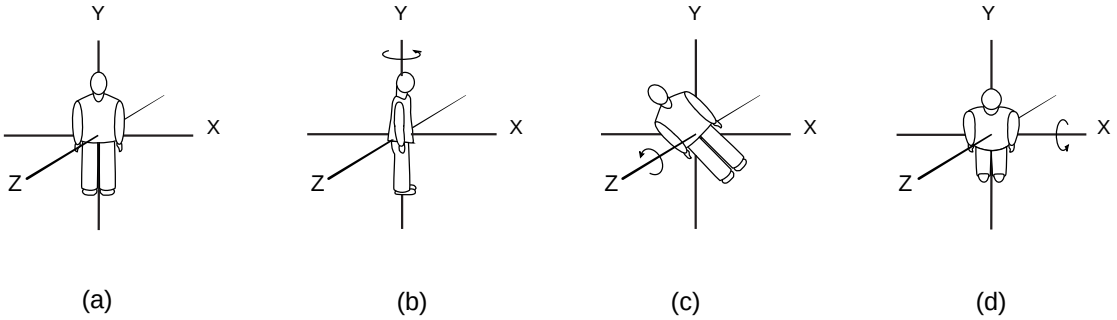


Figure 1-40. Rotations of an object are described as taking place around the axes of the coordinate system.

The third basic transformation is the **scale** transformation, which makes objects bigger or smaller by some factor. (Figure 1-42) As with the other transformations, scales

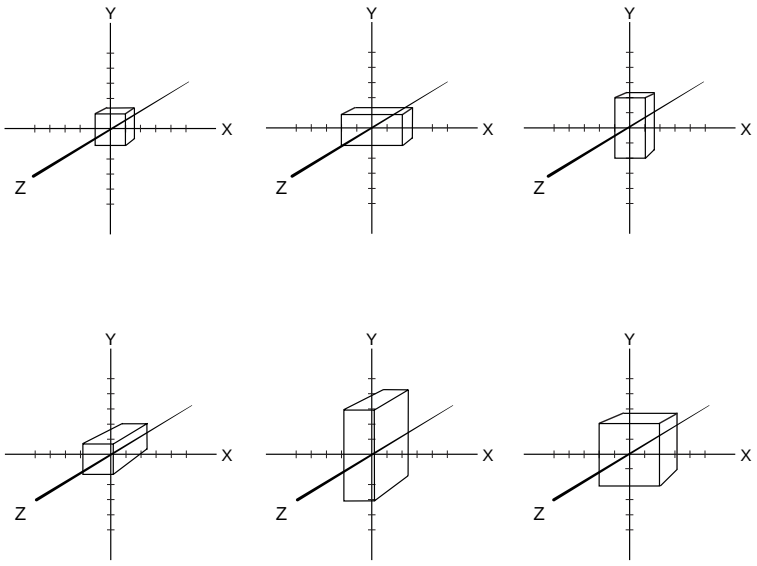


Figure 1-42. Different scaling operations applied to a cube. Scales may be either nonproportional or proportional.

are with respect to each of the three axes. Making an object bigger or smaller in one direction is an x scale, in another direction a y scale, and in the third direction a z scale. Scaling an object equally in all three axes makes an object uniformly bigger or smaller, without changing its proportions. This is called a **uniform scale**, or a **proportional scale**. For example, a uniform scale of two will make an object twice as big. If the scaling is not equal in all three axes, the proportions of the object change and we have a **non-uniform** or **non-proportional scale**.

Each of the three transformations can be effected in one of two ways. If we specify the final result of the transformation, we have an **absolute transformation**. For example, if I say that I want to rotate an object (0,45,0) (that is, 0 degrees around the x axis, 45 degrees around the y axis, and 0 degrees around the z axis) and I mean that I want the object to *end up* in that rotation, then I am making an absolute rotation.

Another possibility, however, is to make a **relative transformation**. A relative transformation specifies the amount that I want to change the object. For example, if I make a relative rotation of (0,45,0), I will rotate the object these additional amounts around each of the three axes. If the object was already at a (0,45,0) rotation and I do a relative rotation of another (0,45,0), it will end up at a rotation of (0,90,0).

The final result of all three transformations in all three axes can be represented as nine numbers, which are often printed as a 3 x 3 grid, or matrix, call the **transformation matrix.** This is illustrated in Figure 1-51. At any given moment, the contents of the transformation matrix describes the combined translation, rotation, and scale of an object.

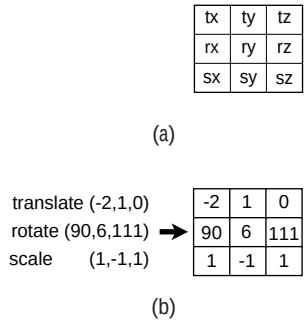


Figure 1-51. The nine transformation values can be organized into a transformation matrix.

Keyframing

Both video and film consist of a sequence of still images, or **frames**. When this sequence of frames is played at the correct rate, we see moving imagery. Animation is the process of creating these frames individually.

The most basic way of animating an object in three dimensional computer graphics is to define the three transformations - move, rotation and scale - of the object at one moment in time and to save that information. This is called a **keyframe**, since it is an important, or key, frame. You then define new transformations for the object at another moment in time and save that information. The whole process just described is called keyframing.

Once an animator has defined several keyframes, the

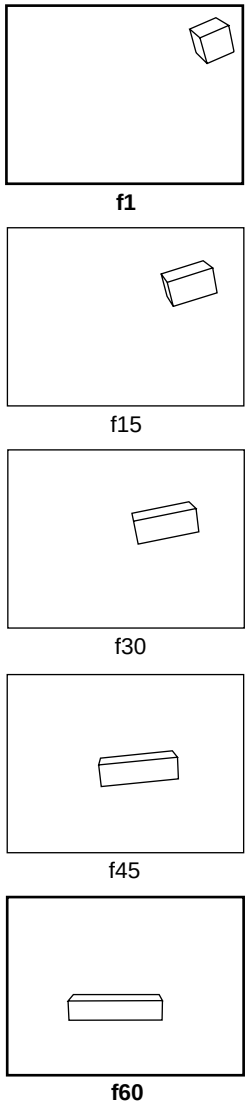


Figure 3-2. A sequence of computer-generated frames. Frames 1 and 60, outlined in bold, are keyframes explicitly defined by the animator. The other, in-between frames were automatically calculated by the computer.

computer can then calculate the transformation values of the object for the in-between frames. This is called **in-betweening**. For example, suppose you position a cube at frame 1 so that it is in the upper right of the screen, slightly tilted and at its normal scale, as in Figure 3-2. You save this as keyframe 1. Then, at frame 60, you reposition the cube so that it has moved down and to the left, has rotated so that it is no longer tilted, and has scaled non-proportionally. This becomes your keyframe 60. With these two keyframes, represented as the top and bottom images in our illustration, the computer can now calculate the in-between frames from 2 through frame 59 by calculating the in-between values of each transformation at each frame. For example, if the translation values at frame 1 were (6,4,0) and are (0,0,0) at frame 60, then the values at in-between frame 30 might be (3,2,0) – that is, halfway between the values of frame 1 and those of frame 60. Similar calculations would be done by the computer to determine the in-between values for rotation and scale. The new in-between values at each frame are called **interpolated values**.

Wireframe Preview

Once the animator has created an animation by defining keyframes, he or she can view it on the computer by requesting an animation playback or **preview**. Here, the computer screen will display the animation by showing, in quick succession, each of the in-between frames. Often, this preview uses a simplified wireframe display of the object, showing only the edges of the object's surfaces, but not the full-color, shaded surfaces themselves. This is equivalent to what traditional animators call a **pencil test**.

The rate at which the animation plays back on the screen can be controlled by the animator. In the U.S., video uses 30 frames per second of animation and film uses 24 frames per second. A ten second video animation, for example, will consist of 300 frames (30 x 10 = 300). By specifying either 30 frames per second or 24 frames per second, the animator can control how quickly the animation will play on the screen. If the animator doesn't specify the playback rate, the computer will usually play back the animation as quickly or slowly as it can, depending on the complexity of the animation scene. This is usually not desirable, since it doesn't allow you to see the movement of your animation at its true speed. A ten second animation may play back too slowly, taking 15 or 16 seconds, or too quickly, taking perhaps only 6 seconds.

Parameter Curve Editing

A very powerful technique for creating and modifying computer animation makes use of a graphic representation of each transformation value. The y scale of an object, for example, might change from 1.0 at frame 1 to 3.0 at frame 100 and then back to 1.0 at frame 200. A graph of this change might look like what we see in Figure 3-13. Each of an object's nine transformation values, or **parameters**, can be graphed in a similar way.

Changing, or editing, the graph of a particular transformation's values automatically changes the animation itself. For example, if you edit the curve of the cone in Figure 3-14 by moving the 3.0 at frame 100 straight down to a value of 2.0, as in Figure 3-13, you will have changed the animation so that the cone now gets a little bit less tall in the same amount of time. Similarly, if you move that same point towards the left from frame 100 to frame 50, you will have speeded up the growth of the cone in the beginning of the animation, since it now takes only 50 frames (less than 2 seconds) to grow to full height, instead of the 100 frames (more than 3 seconds) it originally took. At the same time, you will have slowed down

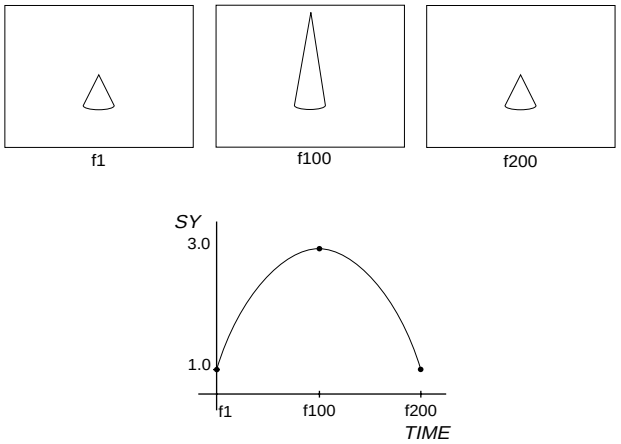


Figure 3-13. The graph of the animation of an object directly corresponds to the actual animation of the object.

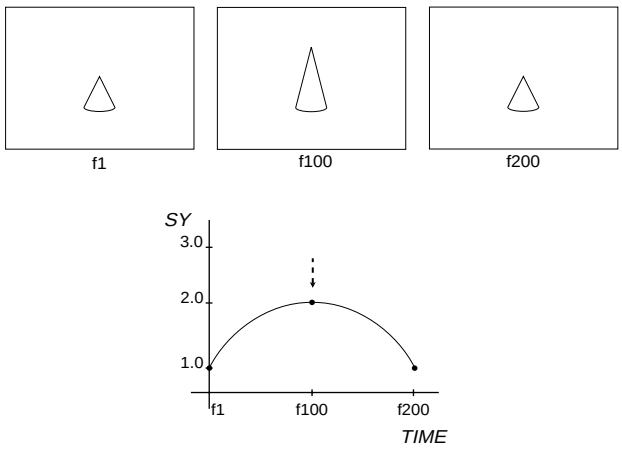


Figure 3-14. By pulling a keyframe control point down along the vertical axis, you change the value of the parameter—in this case, the height of the cone.

the decrease in the cone's height in the latter part of the animation, since that now takes 150 frames instead of the original 100 frames.

It is also possible to change the shape of a parameter curve without changing the actual keyframe values. For example, without changing the keyframe values, we can instruct the computer to calculate different types of interpolation for the in-between frames. Figure 3-6 illustrates several ways of doing this. One type of interpolation is **linear interpolation** and is represented by a straight line. The other types of interpolations illustrated here – Cardinal, B-spline and Bézier – are **spline interpolations**, all of which produce smooth, gradually changing curves. Changing the shape of a curve by changing the interpolation type changes the rate at which the object changes. A curve that is rising or falling very steeply indicates a fast rate of change. A curve that is nearly flat indicates that the object is changing only very slowly.

Let us assume, for example, that the graphs in Figure 3-6 represent changes in the Y scale of the same cone we saw earlier. If we look at the interpolation produced by the Bezier curve, we see that, in the beginning of the animation, the cone is scaling up very rapidly, because the curve is rising very steeply. Around the second keyframe, at the top of the curve, however, the cone's scale is changing very, very slowly, because the curve almost flattens out at that point. Later, at the end of the animation, the scaling changes are once again quite rapid, as indicated by the steeply dropping curve.

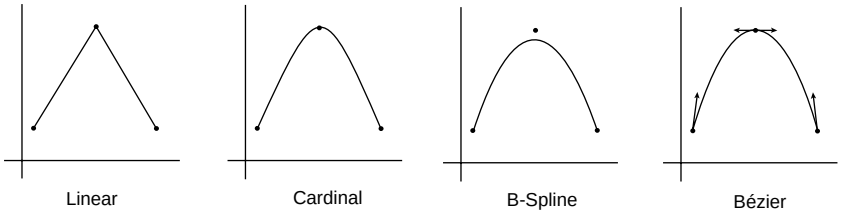


Figure 3-6. For the same set of keyframe values, different types of interpolation yield different animations.

Exercise 1

In this exercise, you will model a few simple primitives, transform them with translation, rotation and scale, save keyframes to make an animation, playback the animation as a wireframe display, and fine-tune the animation by editing its function curves.

1. Make sure you are in the right database

- Click on ***>Get >DB Manager*** (upper-left menu bar). At the very top of the window that opens, you should see: *Chapter in Database <Course34_Practice>:* If you do, this is correct, so just **Exit** this window. If you do not, you need to double-click on the **..** icon. When you see *Course34_Practice,* single-click on it to select it. Then click on Default DB (right menus) to make it the database you will use for these exercises.

2. Build some models

Create a few primitives

- First, use the ***>Get >Primitive >Grid*** command (upper-left menu bar) to model a ground plane. You can use the default settings for the grid.
- Next use the same ***>Get >Primitive*** command (upper-left menu bar) to model three or four primitives. For example, use ***>Get >Primitive >Sphere, >Get >Primitive >Cone***, etc.

Transform your models

- After you've made three or four primitives, select one at a time (but not the grid) by using ***>Select >Toggle Mode*** (lower-right menu bar) and clicking on the primitive you want to select. (You can also just hold down the Space Bar and click on the primitive, as a shortcut way to select it.)
- Once you've selected a primitive, give it some initial configuration by using the ***>Trans, >Rot*** and ***>Scale*** commands (middle-right menu bar). To work in only one axis, click on just that transformation button – for example, ***>Trans X***. To work in all three three axes, click in the border to the right of the transformation buttons.
- Position your primitives so that they are spread out and floating above the grid.

Undoing a mistake

- If you make a mistake, you can use ***>History >Undo*** (lower-left menu bar) to undo the last command.

- If you need to delete an object, select it, then use ***>Delete >Selection*** (upper-left menu bar).

Save your file

- Save your file with ***>Save >Scene*** (upper-left menu bar). Use *Exercise1* as the Prefix and *Prims* as the Scene Name.

3. Animate the objects with keyframing

Define the length of your animation

- Go to the Motion module of SoftImage by clicking on the **Motion** button (in the menu bar across the top-right of your screen).

- First, we want to change the ending frame number to 300, so that we will have a 300 frame animation – that is, a ten second animation. At the far lower right, next to the **E**, change the ending frame number from 100 to 300.

Keyframe the first primitive

- First, make sure the frame counter (the little triangle in the slider bar at the bottom of the screen) is set all the way to the left at frame 1.

- Now, select one of the primitives. Adjust its location, if you like, by using the ***>Trans*** buttons. This will be its starting location.

- When the object is translated as you like, save its current translation as a keyframe at frame 1 by using ***>Save Key >Object >Explicit Translation >All***. (Don't use ***>Keypath Translation*** or ***>Transformations***, please. They create path animation, which we'll deal with in Exercise 3.)

- Now, use the right mouse button to slide the frame counter triangle over to frame 150.

- With the same primitive still selected, use the ***>Trans*** button again to translate it to a new position. When it's where you want it, use ***>Save Key >Object >Explicit Translation >All*** again to save this translation information at keyframe 150.

- Now use the right mouse button again to slide over to frame 300. Use ***>Trans*** again to move your object to a new location. Then use ***>Save Key >Object >Explicit Translation >All*** again to save this translation keyframe.

- Click on the frame counter triangle now and slide it back and forth. You should see your object translating around on the screen as you animated it.

- Save your file again with ***>Save >Scene***. Use the same Prefix and Scene Name you used before. Softimage will save a new file, adding a new version number to it. This new file will not overwrite your old file. (You should save your scene frequently as you work.)

Keyframe the second primitive

- Now select another primitive. Use the mouse to slide back to frame 1.

- On this primitive, we'll animate some rotations. At frame 1, use ***>Rot*** to rotate your object into its initial orientation. Now use ***>Save Key >Object >Rotation >All*** to save this rotation information as keyframe 1.

- Next, use the right mouse button to slide the frame counter over to another frame. Use ***>Rot*** again to rotate your object with the mouse into a new orientation. If you prefer, you can click on the tiny triangle in the upper left of each Rotation button and type in numerical information for your rotation transformation. When your object is rotated as you like, again use ***>Save Key >Object >Rotation >All*** to save the rotation information for this object at this keyframe.

- Use the right mouse button to slide to another frame number, and repeat this process: rotate your object, then save its rotation with ***>Save Key***. (If you click the ***>Save Key*** button with the middle mouse button instead of the left mouse button, Softimage will remember what your last >Save Key operation was and do that for you, without you having to go through all the sub-buttons.)

Keyframe a third primitive

- Repeat this entire procedure for another primitive, this

time using the scale transformation. Select the object. Go to frame 1. Scale the object with ***>Scale***. Save this keyframe with ***>Save Key >Object >Scaling >All.***

- Move to a new frame number. Re-scale the object. Save the new keyframe. Create as many keyframes as you want to for this object.

- Save a new version of your scene, with the same filename – that is, *Exercise1-Prims* – with ***>Save >Scene***.

4. Play back the animation

- To play back the animation segment you just made, click on ***>Play Control*** (lower-right menu bar) and set **Frame Step** = 0. This will force the playback to synchronize to the default 30 frames-per-second rate which is standard for video. Click OK to exit this window.

- Now use the Play button (the blackened triangle pointing to the right) in the extreme lower right of your screen to play the piece of animation you just made.

- To see the animation full-screen, click on the little rectangles icon in the upper right of the Perspective window. To return to all four windows, click again on that same icon.

5. Changing a keyframe

- You can overwrite a keyframe if you don't like your animation. First, select the object. Then move to the frame number you want to overwrite. You can do this either by dragging the frame counter in the slider bar, or by clicking on the current frame number (lower right of the screen) and typing in a new frame number.

- If you don't remember the frame number of your keyframes, turn on the **k** button next to the playback Play triangle. Now click on the Play triangle: the animation will advance to the next keyframe for the selected object.

- With the correct object and frame number selected, re-transform your object, then use ***>Save Key >Object*** with the appropriate transformation to overwrite the old keyframe information.

- You can keyframe more than one transformation for a given object. For example, instead of having each of your primitives either translate or rotate or scale, you could have one of them do all three transformations simultaneously. To do this, simply advance to the desired frame number, transform the object, and then use ***>SaveKey >Object*** with the appropriate transformation.

6. Deleting Animation

- If the animation of one of your objects gets very confused, you can throw away the animation for that object, but still keep the object. First, select the object. Then click on ***>FcrvReset >Object >All*** (lower-left menu bar). You can confirm that the animation is gone by dragging the frame counter triangle. The object should not move at all.

- If your animation gets so confused that you want to start completely over, first use ***>Save >Scene*** to make a backup of your current animation (just in case). Then use ***>Delete >All*** (upper left). This will delete everything on the screen.

- Once you've deleted everything, you can use ***>Get >Scene*** (upper-left menu bar) to retrieve one of the earlier versions of your animation. (This is why it's important to save your scene frequently as you work.)

- After you make any changes, save a new version of your scene, using the same filename.

7. Function curve editing

Open the Function Curve window

- You can fine-tune your animation by editing the function curves. Select the object that you translated, and then use ***>FcrvSelect >Object >All*** to look at the curves for that object. This will open the "Function Curve" window.

- You will see one curve for each of the animated transformations on that object – in this case, Translation X, Translation Y, and Translation Z. The name of each curve is listed in the box to the right. For example, *cube1.etrnx* means "explicit translation in X for the object 'cube1' ".

Moving around in the Function Curve window

- You can zoom in or out of the Function Curve window by holding down the **z** key and then simultaneously holding down either the middle or right mouse buttons. The middle mouse button zooms in; the right mouse button zooms out.
- In order to slide around in the Function Curve window, hold down the **z** key and simultaneously hold the left mouse button down, then drag the mouse cursor.

Editing a keyframe

- Select one of the curves to work on by clicking either on the curve itself or on its name (for example, *cube1.etrnx*) in the upper right corner of the screen.
- Keyframes are displayed as a cluster of three small blue dots. The dot in the middle is the keyframe itself. The two dots on either side of it are the tangent "handles" for that keyframe.
- With one of the curves selected, click on **EDITKEY** at the top of the Function Curve window. Click with the left mouse button on a blue keyframe dot. (Remember, the keyframe is the blue dot in the center.) Still holding down the left mouse button, drag the keyframe dot around. Dragging left-right changes the time of the keyframe; dragging up-down changes its value.
- In order to insert a new keyframe onto the selected curve, click on the curve with the middle mouse button.
- To delete a keyframe, click on it with the right mouse button.
- You can also use the **SETKEY** function from the top of the window to type in numerical information for a particular keyframe. Click on **setkey**, then click on the keyframe. This will open up a window where you can type in numerical information.

Changing the Interpolation of a Curve

- To change the type of interpolation at a keyframe, select **INTERP** from the top of the Function Curve window.

Look at the information bar at the bottom of the window to see what each mouse button does in this mode. Click on a keyframe with the different mouse buttons to get a different type of interpolation at that keyframe.

- You can also change the shape of the curve by selecting **EDITKEY**, and then clicking with the left mouse button on one of a keyframe's tangent handles. Hold the mouse button down and drag one of the handle-dots. As the tangent handle moves, the shape of the curve changes.

- Any changes you make to the function curves immediately affect the animation. You can see your new animation either by dragging the frame counter slider or by clicking the playback triangle to play the animation.

Editing the Curves for a different object

- If you want to edit the curves for a different object, select that object as you normally do with ***>Select***. Now click on ***>FcrvSelect >Object >All*** again. Your Function Curve window will now display the curves for this new object.

Finishing up

- To close the Function Curve window, click with the middle mouse button on the word **Fcurve** in the upper-left of the Function Curve window.

- Make sure you save the most recent version of your scene with ***>Save >Scene***. We will be re-using the scene you created here later in the next exercise, so it's important for you to save the animation you produced in this exercise.

PART 2

Lecture 2

The Camera

Lighting

Surface Characteristics/Shaders

Basic Texture Mapping

Rendering & Shading Algorithms

Final Frame Considerations

Flipbooks

The Camera

In addition to defining the motion of an animation sequence, it is also necessary to define a variety of factors that will determine the final, full-colored look, or **rendering**, of each frame. The first of these factors is the viewer's point of view, or what is called in computer graphics, the **virtual camera**. As in photography or film, the camera in computer graphics consists of several components. The first of these is the **camera location**, or where the viewer is standing. Like any point or object in three dimensional computer graphics, this is specified as a triplet of x, y, z coordinates. The second component of a virtual camera is where the viewer is looking. This is often called the **camera target**. Like the camera location, it is most often represented as an x, y, z triplet, although sometimes it is represented as a rotation of the camera. The third major component of a computer graphic camera is called the field of view and corresponds to the lens of an actual physical camera. In computer graphics, the **field of view** is measured as an angle emanating from the camera location. A wider angle in the field of view corresponds to a wide-angle lens and produces more extreme perspective distortion in the image. A narrower field of view produces little or no perspective distortion. (Figure 2-16)

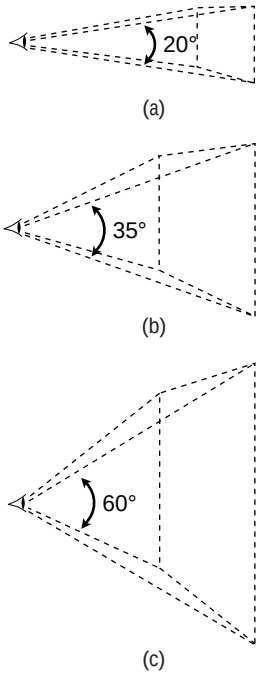


Figure 2-16. Increasing the field-of-view angle of the pyramid of vision produces a zoom-in.

Animating the camera is similar to animating an object. You set the camera location, target and field of view, and save a keyframe for that camera at a given moment in time. You next re-define the camera's components, and save a new keyframe. The computer then automatically calculates the interpolated values of the camera's components, producing camera animation for your sequence. As with any object, you can then playback your animation, re-define some keyframes or edit the parameter curves which represent the camera's animation.

There are several standard camera moves in computer graphics. To move the camera in and out along the depth axis is a **dolly.** To move it left or right or up or down is a **track.** Rotating the camera involves either a **pan**, if the point around which you rotate is the camera itself (that is, the camera location), or a **tumble** if the point about which you rotate is the camera target. (Figure 2-10) A change in the camera's field of view is a **zoom**, as we saw a moment ago in Figure 2-16. Notice that a zoom is not the same as a dolly, even though both will make objects appear larger. When you zoom, the camera does not move in or out – only the field of view changes. When you dolly, you actually move in or out towards or away from the target.

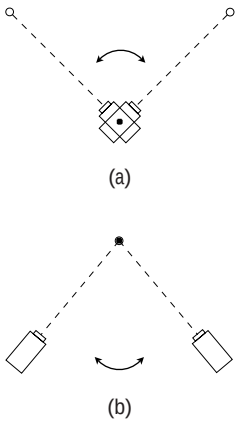


Figure 2-10. A pan and a tumble involve rotations of the camera around different pivot points.

Lighting

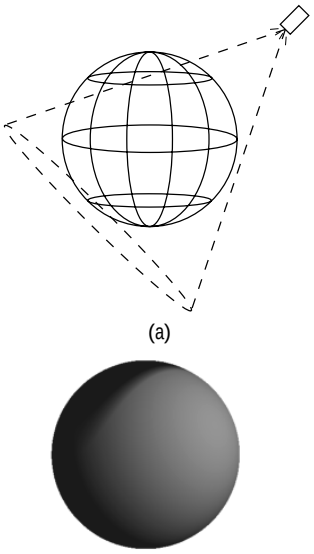


Figure 2-24. Anything outside the cone of the spot light is not illuminated.

In setting up a scene it's also necessary to define lights. A common kind of light in computer graphics is **ambient** light, or ambiance. Ambient light has no specific location but is, rather, the overall amount of light present everywhere in the atmosphere. Another basic type of light is a **point** light, which is similar to a bare lightbulb hanging from a wire. A point light does have a specific x,y,z location and emits light equally in all directions. A **spot** light, like a point light, has an x, y, z location, but emits light only in a specific direction, which is usually controlled by rotating the spot light to aim it at a target. A spot light also has a cone angle which controls the width of the cone of light emitted by the spot. (Figure 2-24) All of these lights also typically have an **intensity** control to govern how bright the light is and a set of color controls for the **color** of the light. Light color is usually defined in terms of the standard red, green and blue (RGB) color components which are so common in computer graphics. Any of the parameters of a light – its location, its color, its intensity, etc. – can be animated, whether by keyframing it, or by editing its parameter curve, or with any of the other techniques used for animating objects in computer graphics.

Surface Characteristics/Shaders

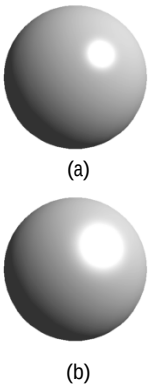


Figure 2-36. Highlight size can be used to fine-tune the surface definition of a shiny object.

In setting up an animation scene, we also have to think about the specific appearance of each surface in the scene. For example, what color should a given object be? Should it be smooth or bumpy? Is it shiny? Is it transparent or opaque? And so on. The set of all these characteristics of a surface is usually known collectively as the **shader** of an object since it is used to produce a full-color, shaded rendering of the object. It is also sometimes known as the **material** of the object. The process of defining a shader involves setting parameters to define the color of the surface, the bumpiness or roughness of the surface, the transparency or opacity of the surface and the shininess of the surface. Shininess usually consists of several components: how bright the highlights are, how large the highlights are and what color the highlights are. (Figure 2-36)

Basic Texture Mapping

Most objects in the real world do not consist of a single, uniform color. Rather, the surfaces of most objects are

multicolored in various ways. When defining a shader of an object, this multicolored character of the surface can be achieved through a very powerful computer graphic technique called **texture mapping.** In the simplest form of this technique, a two dimensional picture, or texture image, is applied to a surface. The specific way in which the texture image applies to the surface is referred to as the mapping of the texture, hence the term texture mapping. One simple way in which an image can be mapped to a surface is to project it onto the surface, as if the image were emitted from a slide projector. This is known as **planar projection mapping**. (Figure 2-55) Another mapping approach is to stretch the image over the surface of the object, as if the image were a sheet of very flexible latex. (Figure 2-58) With this technique, the surface is thought of as having a U direction and a V direction, and the technique is called **UV mapping.**

Producing a texture image that gives you the result you want on your surface can sometimes be difficult. A technique which helps solve this problem is called **3D painting**. In this technique, you are able to use the mouse to "paint" directly on the surface of your three-dimensional computer model. As you do so, the software automatically creates the two-dimensional texture image which corresponds to what you see on your three-dimensional surface.

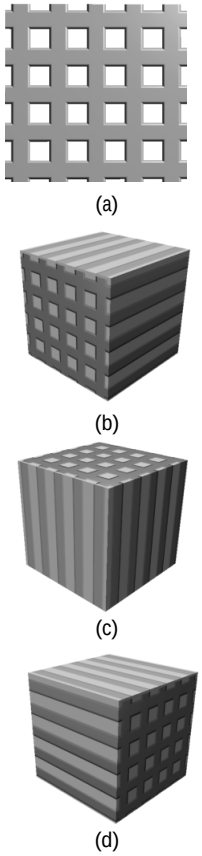


Figure 2-55. Planar projection mapping.

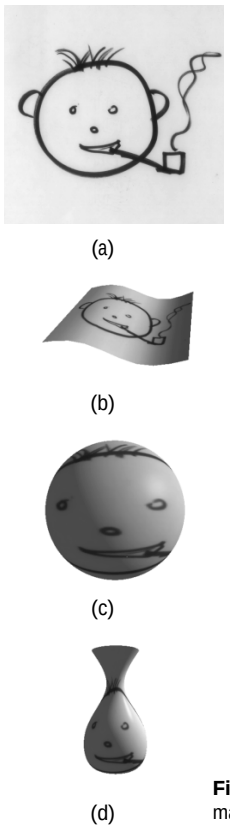


Figure 2-58. Parameterized texture mapping.

Rendering & Shading Algorithms

Even though two scenes might be identical in all the components we've discussed so far – motion, camera, lighting, shading, etc. – the final rendered frames of the two sequences might look quite different from each other. This is because the individual computer programs used to create the rendering might take different approaches to calculating the rendered image. A specific and methodical approach to calculating a rendered image is called a rendering algorithm.

One common rendering algorithm is **raycasting**. In this algorithm, the program calculates the trajectory of a single imaginary beam of light as it passes from the eye through each pixel of the screen and into the computer graphic scene. Wherever this beam hits a surface, the color of the surface at that location is calculated and that color becomes the color of the pixel. (Figure 2-45)

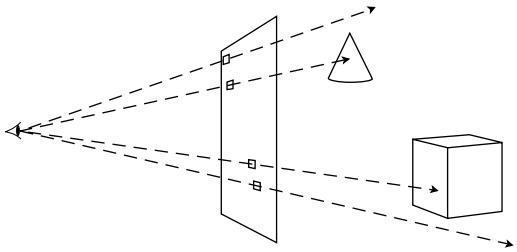


Figure 2-45. The ray casting algorithm casts a single ray through each pixel. If the ray strikes a surface, the color of the surface is calculated and assigned to the pixel.

A more refined rendering algorithm is **raytracing**. In this algorithm, each beam of light which is sent through a pixel is traced as it bounces among the scene's reflective surfaces and passes through any transparent surfaces. (Figure 2-53)

In addition to its rendering algorithms, each program also uses a set of **shading algorithms**. A shading algorithm is a particular technique for calculating the color of a surface once the rendering algorithm has determined how much light is hitting the surface. The most common shading algorithms – Lambert, Gouraud, Phong, and Blinn shading –are named after their inventors. Each has certain characteristics and certain advantages and disadvantages. In setting up a rendering, you

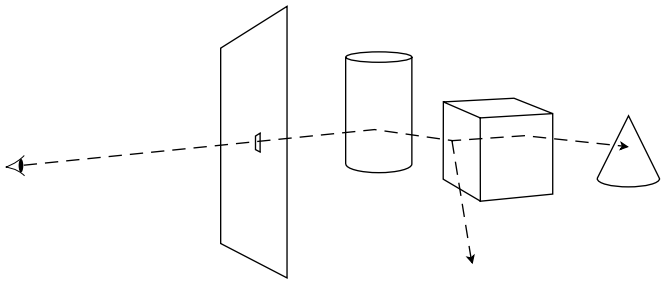


Figure 2-53. The ray-tracing algorithm traces a light ray back as it bounces off objects in the scene.

must select both the rendering algorithm and the shading algorithms you wish to use.

Final Frame Considerations

There remain a number of other factors to be considered before rendering your final frames. One of these is the proportions of the image - that is, its width relative to its height. This is called the **aspect ratio** of the picture. Rendering for film may involve selecting one of several standard film aspect ratios. The standard aspect ratio for video in the U.S. is 4/3.

Another final frame consideration is the actual number of pixels in both the horizontal and vertical dimensions. This is the **resolution** of the image. A typical image resolution for video in the US is 640 x 480 pixels. (Notice that this adheres to the 4/3 aspect ratio.)

Another final frame consideration is how carefully you want the rendering program to do calculations to smooth out the rough edges, or jaggies, of surfaces. This process is known as **anti-aliasing**. (Figure 2-97)

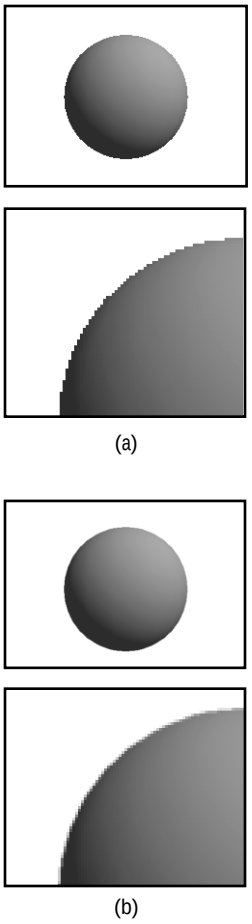


Figure 2-97. Aliasing, or the "jaggies," can be overcome by activating anti-aliasing calculations in the final rendering.

Flipbooks

Once all the frames of a sequence have been rendered as full-color images, they can be played back as a moving animation on the screen. This is usually known as a **flipbook**. As with the wireframe preview which we described in Lecture 1, it is important to specify the **frame rate** for the flipbook playback so that it will play at the intended speed.

Exercise 2

In this exercise, you will define a camera and some lighting for the scene you produced in Exercise1. You will then define materials and textures for your models. Finally, you will render frames for your animation and view your frames as a flipbook.

1. Setting up your camera

Define the camera

- Click on ***Matter*** in the top menu bar to go to the Matter module of Softimage. Retrieve the last version of your *Exercise1-Prims* file with ***>Get >Scene***.
- Click on ***>Camera >Picture Format*** (top-right menu bar) and next to **Video** select the **NTSC** video format for your animation frames.
- Now use ***>Camera >Orbit, >Camera >Dolly***, and ***>Camera >Tracking*** (upper right) to position your camera in a way that looks good for your animation. Pay attention to the message bar (bottom of the screen) which tells you what each mouse button does for each camera move. (Do not attempt to animate the camera yet; we will do that in Exercise 3.)
- Play back your animation with this camera to make sure it looks ok.

View your scene in Shaded mode

- In the bar above the Perspective window, click on the word **SHADE** and click again on **Shade** in the pull-down menu. This changes the display of that window to a real-time shaded image.
- In the far upper-right of the Perspective window , click on the rectangles icon to make the Perspective window full-screen. Try playing back your animation again. Click on the rectangles icon again to go back to the normal, four-window display.
- Use ***>Save >Scene*** to save your scene. Use *Exercise2* as the Prefix and *Render* as the Scene Name.

2. Setting up your lights

Define a point light

- Click on ***>Light >Define*** (upper right) to create a light. Under **Type**, make sure **Point** is selected. Click on OK.

This will create a default point light at 0,0,0. The light is represented by a lightbulb icon.

- Back in the normal Softimage window, use ***>Trans*** to move this point light wherever you like. You will see the effects of your lighting changes in the shaded rendering of the Perspective window.

- Now try editing the parameters of the light. With the point light still selected, click on ***>Light >Edit*** (upper right). Inside the light editing window, make some changes. For example, you can make the light brighter or darker or change its color by adjusting the **RGB** sliders (upper right). Click OK to accept the changes.

Define a spotlight

- Use ***>Light >Define*** again, and this time change the **Type** to **Spot.** Adjust **Cone Angle** to about 60 degrees (middle right). Under **Shadows Type** (lower left), select **Soft**, so that the spotlight will create soft shadows. Click OK to accept all this.

- Use **>Trans** to move the spotlight where you want it.

Preview the rendering

- Notice that the rendering you see in the Perspective window shows no shadows, even though you just turned them on. This is because the SHADE rendering mode we turned on for this window is very crude – fast, but crude.

- In order to see a better quality rendering, use ***>Preview >All*** (lower left). This will render a high-quality rendering of your frame.

- Click the middle mouse button to remove this rendering.

- Move your frame counter to another frame of your animation and use ***>Preview >All*** again to see what a rendering of that frame looks like. If necessary, adjust either your point or your spot light, whether by moving them with ***>Trans*** or by changing their parameters with ***>Light >Edit.***

- Save a new version of your scene under the same name, *Exercise2-Render,* with ***>Save >Scene***.

3. Defining Surface materials

Define a shiny surface for one object

- Select one of your primtives and click on ***>Material*** (middle left).
- Inside the Material Editor window, make sure the **Shading Model** (top left) is **Phong** .
- Adjust **Specular Decay** to change the size of the highlight.
- To define the surface's color, you work with the color triangle (left middle). **Diffuse** is already selected. Use the **RGB** sliders to change the Diffuse color of the object.
- When you have something you like, select the **Ambient** box of the triangle by clicking it with your left mouse button. We will begin by copying the Diffuse color to the Ambient box. With the Ambient box selected, click on the Diffuse box with your right mouse button. This gives Ambient the same color as Diffuse.
- In order to make Ambient a darker shade of the color you just gave it, click on **RBG** (just below Ambient) to change it to **HLS** (Hue, Luminance, Saturation). Drag the **L** slider to the left.
- If you want to change the color of the highlight, click on the **Specular** box at the top of the triangle and change its color.
- You can preview a rendering of this material on your object without exiting the Material Editor window. Click on **Preview** in the lower left of the Material Editor window. Use the middle mouse button to remove this image.
- When you have a material you like, click on OK to apply it to your selected object.

Define a reflective surface for another object

- Select one of your objects which is nicely curved (for example, a sphere). Click ***>Material*** as before. Follow the same procedure to define the color of the object.

- Now, drag the **Reflectivity** slider (middle right) to the right. The pattern you suddenly see on the demo-sphere is just to show you how much you've increased Reflectivity; it's not really part of your material.

- Still inside the Material Editor window, click on **Preview** to see a rendering of this material on your object. Notice that you see the correct color, but you see no reflections. This is because the Preview function inside the Material Editor renders only the selected object. Since no other objects are rendered, there is nothing to reflect on the sphere. Click the middle mouse button to remove the Preview picture.

- Back in the Material Editor window, click **OK** to apply this material to your object.

- Back in the normal Softimage window, click ***>Preview >All.*** Since this renders all the objects in the scene, you should now see some reflections in the object you just made reflective.

Define surface materials for the other objects

- Follow a similar procedure for each of your other floating primtives. (Skip the grid for now.) Select the object and use ***>Material*** to define that object's material. Try giving one of your objects some transparency by sliding the **Transparency** slider bar up. Also, try changing the **Shading Model** to **Lambert** to remove all highlights.

Put a texture on the grid

- Select the grid which is under the floating primitives. With the grid selected, use ***>Texture >2D Global*** (middle left).

- At the top of the window that opens and just to the left of **Picture Filename**, click on **Select** to select a texture picture. When the window opens up, double-click on the **..** icon to go up one directory level. Do this again to go up another directory level.

- Double-click on the **SI_material_lib** folder to go into it. Inside that, double-click on the **PICTURES** folder, and inside that double-click on the **TILINGS** folder. Double-click on the *check* file to select that as your texture picture.

- Back in the Texture window, you should see a black and white checkerboard picture in the center of your window. Just below that, under **Mapping Method**, click on *XY Coordinates.* In the pull down menu that opens up, select **XZ Coordinates**, to project the checkerboard picture down from above onto the grid.

- Click **Preview** (bottom left) to see what this looks like. Remove the Preview rendering by clicking the middle mouse button.

- Back in the 2D Texture window, change **Repeats** (middle left) to 4 by 4. Click **Preview** again to see how this has changed the mapping of the checkerboard picture. Click **OK** to apply this texture mapping to your grid.

Fine-tune your materials

- Back in the Softimage window, use ***>Preview >All*** to see a rendering of your entire scene. Slide the frame counter triangle to another frame and render again. Do this for several frames throughout the animation. If necessary, adjust your materials and/or your lights.

- Save a new version of your scene under the same name *Exercise2-Render,* with ***>Save >Scene.***

4. Render your frames

- Make sure you are still in the Matter module of Softimage. Click on ***>Render*** (lower left).

- Inside the Render Setup window, change the **Resolution** to 320x240, so the frames of our test will be small and fast. (You can reset it to the full-size 640x480 later, once you have gotten successful tests.)

- Under **Sequence**, set **Start** = 1, **End** = 300, and **Step** = 5. This means you will be rendering frames 1 through 300, but only rendering every 5th frame (again, just to make our tests go faster.)

- Look at the bottom of the Render Setup window, just above the word *default*. It should say, *Rendering in Database <Course34_Practice>:* If it does not, click the **DB List** button just to the left and select *Course34_Practice.*

- Change the Filename from *default* to *exercise2*.
- Click on **Render Sequence** to start the rendering. The screen will clear and you will see each frame rendered, one by one. This will take several minutes (which is why we are using a low resolution and only doing every fifth frame for now.)
- If there is something wrong and you need to stop the rendering, click on all three mouse buttons simultaneously.

5. Play back a flipbook

- Click on **Tools** in the top menu bar to go to the Tools module.
- Inside that module, select ***>Flipbook***. (middle left).
- Softimage may not properly remember which database you rendered your pictures into. If you do not see a list of the pictures you rendered, double-click on the **..** icon to go up one directory level. Keep doing this until you find your database. Double-click on it, then double-click on the sub-directory called RENDER_PICTURES.
- Once you have found your pictures, click on any one of the frames you just rendered. (The frame number of the file doesn't matter.)
- Inside the window that opens up, next to **Do frames**, set your **Start, End** and **Step** numbers to 1, 300, 5, just as they were in your Render Setup window. In the **Number of Frames per Second** box, type in 6. (Because 6 x 5 = 30, which is our video playback rate.) Click OK.
- It will take a few seconds for the frames to be read into memory, and then a window will open with your first frame in it. Use the normal animation playback buttons to playback the flipbook animation in this window. For example, in the far lower right, click on **L** to make the animation loop (that is, repeat indefinitely) and then the right-facing triangle to play it.
- Click the middle mouse button to pause the flipbook playback. You can also drag the frame counter triangle to move through the flipbook animation that way.

- Click all three mouse buttons simultaneously to remove the flipbook window.

PART 3

Lecture 3

Polygonal Modeling

Patch Modeling

Common Modeling Techniques

Surface Editing

Keyshape Animation

Object Path Animation

Camera Path Animation

Bump and Transparency Mapping

Almost all three dimensional computer animation programs create surface models. Objects modeled in this way are not solid and have nothing inside (unless we specifically model other objects inside them.) Rather, objects modeled in this way are hollow shells composed of exterior surfaces only. In developing **surface models** for an animation scene, there are two broad approaches.

Polygonal Modeling

The first approach restricts itself to using only flat surfaces. Typically, an object modeled this way would consist of a great many flat surfaces connected together. Each flat surface is called a **polygon**, a word which means "many-sided" in Greek. Each polygon is defined by the points, also called vertices (singular, **vertex**), at its corners, and by the **edges** which connect these vertices.

For those objects which are in fact composed only of flat surfaces – objects such as walls, boxes, tables – polygonal modeling is ideal. For objects with curved surfaces, however, polygonal modeling must use a great many polygons to approximate the curvature of a surface. This is called **polygonal approximation**. The more polygons you use and the smaller the polygons are, the more accurately they will approximate the curved surface. (Figure 1-5)

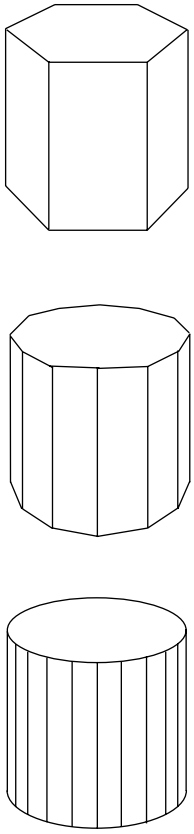


Figure 1-5. A cylinder modeled with three different resolutions of polygonal approximation.

Patch Modeling

Rather than merely approximating the curvature of surfaces, it is also possible to create surfaces which are truly, mathematically curved. Such surfaces are called patches and are derived from curves called **splines**. (Figure 1-23) The mathematics of spline curves is such that they are truly curved and not merely approximations of curves. Consequently, the patches derived from splines are also truly curved.

A big advantage of patches over polygons is that, since patches are truly curved, you can move in as close as you want to a curved patch and never see any straight edges. On a polygon model, by contrast, the individual approximating polygons which make up the surface will become visible if you move in close enough.

On the other hand, a patch model has a specific disadvantage. A patch surface is generated from two spline curves, with each spline being defined by a fixed number of

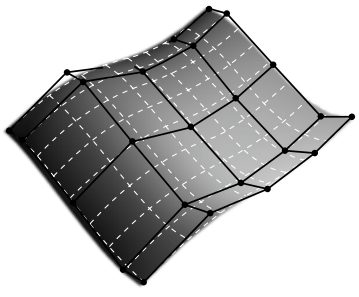


Figure 1-23. The control points of the generating curves create a network of control points, which in turn define the patch.

control points. This means that the resulting patch must consist of a grid, or **mesh**, of control points - a certain number of control points in one direction and a certain number of control points in the other direction. The structure of this control point mesh can be very limiting. Irregularly shaped objects -– such as a human hand with all its five fingers, or a human face with eyes, nostrils and mouth – are extremely difficult to model with patches. Polygonal models, by contrast, are not required to adhere to a regular mesh structure. Polygons can be attached to each other edge-to-edge in any order we choose. This gives them a great advantage over patches in the modeling of certain irregular shapes.

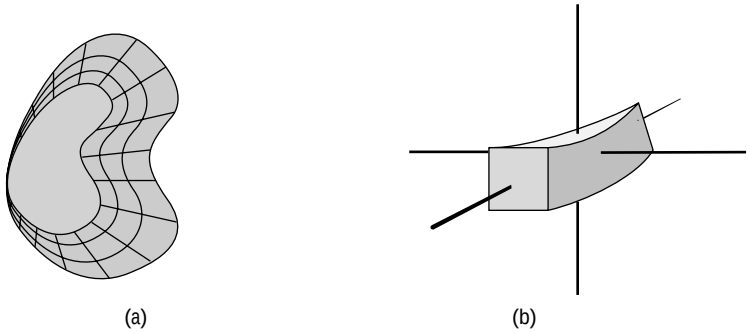
Common Modeling Techniques

There are a number of very powerful modeling techniques that are commonly found in most three dimensional computer graphic systems. One is **extrusion**, in which a contour is pushed, or extruded, along a path. As the contour moves down the path, it traces a surface. The path of an extrusion may be either straight or curved. If the contour is either rotated or scaled as it moves down the path, it is called a **sweep,** or swept surface. (Figure 1-58)

Another common technique is to rotate a contour around a circular path. This produces a surface of revolution. Another technique involves drawing a series of contours and then connecting them together, forming a "**lofted**" surface. Yet another technique involves drawing several curves to represent the edges of a surface, and then connecting them. The surface that results from this technique is called a **boundary** surface.

Surface Editing

In addition to the modeling techniques just described, there are many other modeling techniques available in three



(a) (b)

Figure 1-58. Swept surfaces generated by transforming curves as they are extruded through space.

dimensional software packages, with each package offering its own particular set of techniques and its own particular strengths. One general approach that all techniques share is based on the fact that any surface model in computer graphics, whether a polygon model or a patch model, uses points to define it surfaces. In a polygon model, these points are the **vertices** at the corner of each polygon. In a patch model, the points which define the surface are called **control points**. By moving either the individual vertices of a polygon model or the control points of a patch model, you can change the shape of the surface (Figure 1-24). All the different modeling techniques that a 3D software package offers are, at root, different ways of moving either vertices or control points.

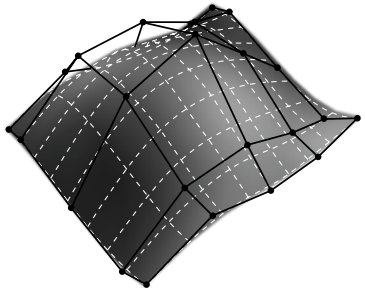


Figure 1-24. Moving the control points of a patch changes the shape of the patch.

Keyshape Animation

A common technique for animating shape changes of a model is to define and save different shapes for it at different frames. These shapes are called **keyshapes.** Once they are defined at each keyframe, the software creates interpolated shapes at the in-between frames (Figure 3-59).

An important characteristic of the keyshape technique is that each keyshape must have the exact same number of points defining its surface. This is because the interpolation that is calculated is an interpolation of the x, y, z coordinates of each point on one keyshape surface to the x, y, z coordinates of the corresponding point on the next keyshape surface. If the number of points on two keyshape surfaces differs, the software doesn't know which point of shape A should interpololate to which point of shape B.

An easy way to assure that all of your keyshapes have the same number of points is to begin by making identical copies of your original model. Once the copies are made, you can modify each copy, changing it into the desired shape.

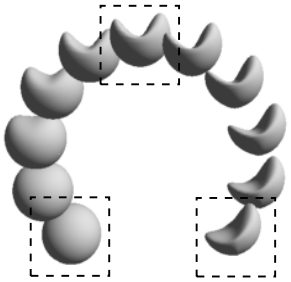


Figure 3-59. The keyshape technique applied to a sphere. Each of the keyshapes is outlined; between these three shapes are the in-between shapes generated by the computer.

Object Path Animation

One way of moving an object through space is to draw the intended **path** of an object and then move the object along that path. This is **object path animation.** Once the object has been animated along the path, you can then change the shape of the path to change the animation of the object. It is also possible to change the timing of the object's motion by editing a separate curve called a **timing curve**. (Figure 3-54)

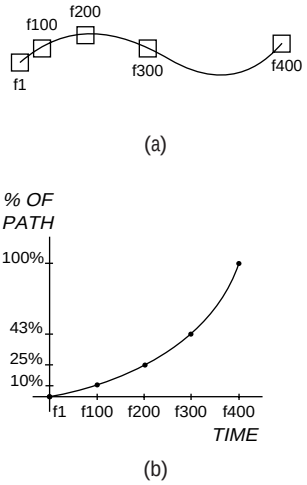


Figure 3-54. If the interpolation of the timing curve is changed, the rate of movement of the object along the path is changed.

This curve represents the percentage of the path covered at any given moment in time, and can be edited in all the same ways that a normal parameter curve can be edited. By changing the shape of the timing curve, you change the speed with which the object moves along the path.

Camera Path Animation

Just as you can animate an object along a path, you can also animate a camera along a path. The approach is similar. First you draw the intended path of the camera. Then you move the camera along the path. As with object paths, there is also a timing curve which can be edited.

Camera path animation is more complex than object path animation, however, because a computer graphic camera is actually composed of several components. In addition to moving the **location** of the camera along the path, you must also move the camera's **target** - that is, what the camera is looking at - along the path. You can even have a separate path for the camera target. Finally, in animating a camera, you must also think of the camera **roll**, or tilt, which is a rotation of the camera about the axis of its line of sight. (Figure 3-69) A roll of 180 degrees, for example, will put the camera upside down, as if you were standing on your head.

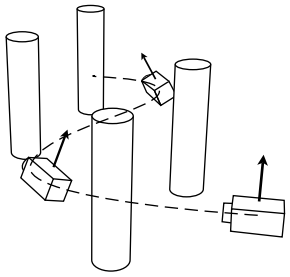


Figure 3-69. The tilt of the camera can also be animated to produce a banking effect.

Bump and Transparency Mapping

In addition to the mapping of colors as described in Lecture 2, it is also possible to use the texture mapping technique we saw there for other effects. One is to create the appearance of bumps, or roughness, on a surface. This is **bump mapping**. The mapping principles are the same. A two dimensional picture serves as the texture image. This image is then mapped, or applied, to the surface, either with projection mapping or UV mapping. Here, however, the color texture mapping and bump texture mapping techniques diverge. In standard color texture mapping, the colors of the texture image map to the surface, producing a pattern of color on the surface of the object. In bump mapping, however it is not the colors of the texture image, but rather the pattern of brightness of the texture image which maps to the surface. This pattern of brightness is interpreted in a certain way to yield a pattern of bumpiness on the surface. (Figure 2-69) Since color information is ignored in a bump texture, bump texture images

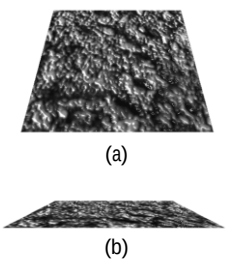
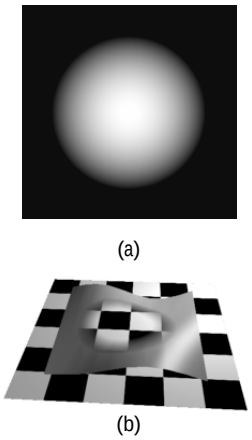


Figure 2-69. Bump mapping does not change the underlying geometry, as the straight edges of the surface show.

are usually black and white. If, for example, the texture image is white in a specific area, then the corresponding area of the textured surface will have a large, prominent bump. If the texture image is gray in that area, the surface's bump will be less pronounced. And if the texture image is black in that area, the surface will have no bump at all in the corresponding area.

A similar technique is used to create irregular patterns of transparency on a surface. White areas on the texture image produce transparent areas on the surface, black areas on the texture produce opaque areas on the surface and gray areas on the texture produce semi-transparent areas on the surface. This is called **transparency mapping.** (Figure 2-66)



(a)

(b)

Figure 2-66. Transparency mapping.

Exercise 3

In this exercise, you will model a terrain and define a bump map for it. You will then place an object over the terrain and create a shape animation for this object. Finally, you will draw a path over the terrain for your camera and move your camera along that path.

1. Create a terrain

Model the geometry of the terrain

- In the Model module of Softimage, use ***>Get >Primitive >Grid*** to make a B-Spline Patch grid. Use settings of 5x5 for the Cell Size and 10 x10 for the Cell Count.

- Holding down the Shift key, hit Shift-A to change all three of the orthographic window to display all of the scene. In the Perspective window, use ***>Camera >Dolly*** (or just hold down the **p** key) to dolly out and make all of the grid visible in that window. Use ***>Camera >Orbit*** (or just hold down the o key) to rotate your camera in the Perspective window.

- We will now use a procedure to randomly move some of the grid's control points to deform it. With the grid still selected, use ***>Effects >Randomize*** (middle-left menu bar). In the option window that opens, change the x, y and z **Size** values to 1 and the **Repetition** to 10, and click OK.

- Save a new version of your scene with ***>Save >Save Scene.*** Use *Exercise3* as the Prefix and *Terrain* as the Scene Name. Give the terrain a material and texture

- Click **Matter** (top bar) to go to the Matter module of SoftImage.

- Make sure the terrain is still selected and choose ***>Material***. Select Lambert as a **Shading Model**, so your terrain will not be shiny. Use the **RGB** sliders to create a brownish color. Values of RGB = 0.7, 0.5, 0.3 give you a good start. Click OK to assign this material to your terrain.

- With the terrain still selected, go to ***>Texture >2D Global***. At the top of the window, just to the left of 2D Texture, click Select. In the right side of the window that opens up, click Options and **Use Icons** to display icons for each file, rather than text. Double click on the **..** folder to go up one directory level. Double click on *SI_material_lib* to go into the Softimage material library. Select *bump_heavy*, or something like it, then click Load.

- Back in the 2D Texture File window, under **Mapping Method** (just below the picture), select XZ coordinates to project

the bump texture straight down onto the terrain from above. Click **Preview** (bottom left) to see a preview rendering of your terrain. Hit the middle mouse button to remove the Preview picture.

- Make adjustments to the texture parameters to improve your texture. For example, slide the **Roughness** slider (center bottom) to about 5 to make the bumps lower. To make the bumps smaller in width, change the **Repeat** values (center left) to about 4 and 4. Use **Preview** again to render the terrain. When you have something you like, assign this texture to the grid by clicking OK.

- Save a new version of your scene with ***>Save >Save Scene***, using the same filename, *Exercise3-Terrain.* When you are asked to copy or leave the pict file, select **Leave**. This will leave the *bump_heavy* picture you are using as a bump map in the *SI_material_lib* folder.

2. Create a Shape Animation of a sphere

Model a sphere

- Go to **Model.** Use ***>Get >Primitive >Sphere*** to create a sphere. Make it a Cubic Nurbs sphere.

- Activate all three axes of ***>Scale*** by clicking on the border to the right of the Scale transformations. Then hold down all three mouse buttons simultaneously and drag to uniformly scale the sphere smaller.

- User ***>Trans Y*** to translate the sphere up so that it is hovering above the terrain.

Make copies of the sphere

- Make sure the sphere is still selected. Use ***>Duplicate >Repetition.*** Inside the window that opens, set **No. of occurences** to 3. Under **Transformations**, set the X translation to 5. Click OK. This will create 3 copies of the original sphere, each offset to the right by 5.

- Hold down Shift-z and use the cursor to draw a rectangle around all four of your spheres to zoom in on just those four spheres. Do this in each of the orthographic windows.

Change the shape of the copied spheres

- Now we will change the shape of the copied spheres, but leave the original sphere untouched.
- Select the sphere just to the right of your original sphere. If the points of the sphere are not visible, use ***>Show >Point*** to make them visible.
- With the sphere selected, click ***>Edit >Move Point***, then click and drag on one of the points of the sphere in any one of the orthographic windows. (You could also just hold down the **m** key and simultaneously drag a point.) Do this for several points on the sphere to give it bumps and hollows.
- If you need to get a closer view of your sphere, make sure it is still selected and press the Shift and the **f** keys simultaneously, to frame the sphere within the three orthographic windows.
- Select the next sphere to the right and do the same thing: move the points of the sphere with ***>Edit >Move Point*** (or with the **m** key). Make the shape of this sphere distinctly different from that of the previous sphere.
- Do the same thing to the last sphere. Select it, and use ***>Edit >Move Point*** (or the **m** key) to move some of its points about to change its shape.

Define Key Shapes for the Sphere

- Click on **Motion** (top bar) to go into the Motion module of Softimage. Change the ending frame number to 300, next to the **E** in the lower-right of the screen.
- Use ***>Select >Clear*** to unselect everything. Zoom out of the orthographic windows using ***>Camera > Zoom,*** or the **z** key so that you can see all four of the spheres. Now select the original, undeformed sphere.
- Set your frame counter to 1. With the original sphere selected, use ***>SaveKey >Object >Shape*** (lower-right menu bar) to save that shape of the sphere at frame 1.
- Advance your frame counter to frame 100. With the

original sphere still selected, use ***>Shape >Select Key Shape*** (middle-left menu bar), and then click on the first copied sphere which you deformed. The original sphere will change into this shape.

- Advance your frame counter to frame 200. With the newly deformed sphere still selected, use ***>Shape >Select Key* Shape** again, this time clicking on your second deformed sphere. The leftmost sphere will change into this shape.

- Advance to frame 300. Use ***>Shape >Select Key Shape*** again and this time click on your last deformed sphere. The leftmost sphere will once again change into this shape.

View the animation

- Drag the frame counter left and right. You should see the leftmost sphere animate smoothly from one shape to another. Try zooming in closer to the deforming sphere, and turn on SHADE mode in one of your windows. Drag the frame counter again.

Deleting shape animation

- If the shape animation for your sphere gets very confused, you can delete the shape animation, but leave the sphere. Make sure the leftmost, animated sphere is selected. Set the frame counter to frame 1. Now use ***>FcrvReset >Object > >Shape***.

- Save your scene with the same filename, using ***>Save >Scene.***

- Once you have a good shape animation, you no longer need the copied spheres. You can either make them invisible with ***>Display >Hide >Toggle_Desel. Hidden*** (upper-left menu bar), or you can even delete them entirely with ***>Delete >Selection.***

3. Set up a rendering of your animation

Define some lights

- Go to **Matter.** Use ***>Light >Define >Infinite*** (upper-right) to create a light which simulates light coming from the sun. Use ***>Trans*** to position this light. The distance of the light icon from the grid is not significant, because the light is at an infinite distance – only the direction matters. Placing the light high overhead will simulate a mid-day light. Placing the light to a low position will simulate sunrise or sunset.
- Test your lighting and your texture by moving your frame counter to different frames and using ***>Preview >All***.

Define a material for the sphere

- Click on **Matter** (or the F4 key) to go the matter module.
- With the animated sphere selected, click ***>Material***. Make the sphere reflective by dragging the **Reflectivity** slider (upper right). Leave the color of the sphere a pale gray. Click OK.
- Use ***>Preview >All*** to see a rendering of your scene. You should see the terrain reflected in the sphere. Advance your frame counter to another frame and use ***>Preview >All*** again to see what your animation looks like at that frame.
- Adjust your camera in the Perspective window to give you a good view of your scene. Use ***>Camera >Dolly,*** or ***>Camera >Orbit,*** or ***>Camera >Tracking*** as necessary.
- Save a new version of your scene under the same filename with ***>Save >Scene.***

Render and view your animation

- Still inside the Matter module, use ***>Render*** to create test frames of your animation. Set the frame count to 1 to 300 by 5, and the resolution to 320 x 240, just as we did in Exercise2, for fast testing. Set the filename to *terrain*. Click **Render Sequence**. If you need to interrupt your rendering, click the middle mouse button.

- When the frames are finished rendering, go to the **Tools** module and use ***>Flipbook*** to view the rendered playback. Set **Do frames** 1 to 300 by 5 – the same as what you rendered. Don't forget to also set the **Number of Frames per second** to 6, in order to get a true, 30 frames-per-second playback of your animation.

4. Create a Camera Path Animation

Place some columns on the terrain

- Go the the **Model** module of Softimage.
- Use ***>Get >Primitive >Cylinder*** to create a default cylinder. Use ***>Trans*** to move it away from the exact center of the grid, but not too close to the edge of the grid.
- With the cylinder still selected, use ***>Duplicate >Immediate*** to create a copy of the cylinder. Initially, you won't be able to see it, because it's in exactly the same place as the original cylinder. Use ***>Trans*** to move this new cylinder to a different location, again keeping it away from the edges of your grid.
- Use ***>Duplicate*** again to make several more cylinders, keeping each of them away from the exact center of the grid and not too close to the edges.
- Go to the Matter module and use ***>Material*** to give each column a different color.
- Save a new version of your work, under the same file name, with ***>Save >Scene.***

Draw a path for the camera

- Now we will draw the path that our camera will follow over the terrain. The path will move amongst the cylinders and end with the camera looking at the deforming sphere. Go back to the **Model** module
- Click on the Top window and use ***>Camera >Zoom*** and the middle mouse button to zoom in on the columns. If you need to slide left or right, drag with the left mouse button. Do the same thing for the other two orthographic windows.

- Clean up your window displays by clicking on the right-angle ruler at the top of one of the orthographic windows. Inside the Layout window that opens up, turn off **Grid Visible**, and then click **All Views** at the bottom of the window.

- To draw the camera path, make sure you are in the **Model** module and use ***>Draw >Curve >Bezier*** (middle left) to draw a curve in the Top window. As you click each point, hold down the mouse button and drag to create that point's tangent, or "handle". Draw your curve so that it winds in amongst the columns and ends pointing toward the keyshaped object in the center.

- With the whole curve selected, use ***>Trans Y*** to move this curve up a bit so that it is hovering just above the level of the terrain.

- Now use ***>Edit >Move Point*** (middle left) to move some of the points on your path to make the path bend up and down. You can also use this same function to drag the ends of the curve's tangent handles to change its curvature in that area.

- If you need to delete a point, use ***>Edit >Delete Point.*** If you need to add a point, use ***>Edit >Add Point***, paying attention to the different mouse button functions as described in the information bar at the bottom of the screen.

- Save a new version of your scene with ***>Save >Save Scene.***

Move the camera along the path

- Click on **Motion** to go to the Motion module of Softimage. Make sure the ending frame number is still set to 300.

- Use ***>Camera >Picture Format*** (upper right) and change to an **NTSC** video format.

- Use ***>Camera >Show Camera*** (upper right) to display the camera icon. This may be off-screen, so use Shift-A to make the orthographic windows display all objects. You should now see the camera icon.

- Use ***>Select >Clear*** to make sure nothing is selected.

Now use ***>Camera >Select Camera***. With your camera selected, use ***>Path >Pick Path*** (middle-left menu bar) and click on the path curve you drew. Accept the default frame numbers of 1 to 300.

- Now drag the frame counter or click on the play triangle to play your animation. Notice that while the camera position moves along the path, the camera interest remains fixed in the center of the world at 0,0,0.

Move the camera's interest along the path

- To make the camera look down the path as it moves along, use ***>Camera >Select Interest*** to unselect the camera position and select the camera's interest point. Use ***>Path >Pick Path*** again, again clicking on the path curve. This time make the animation go from frame -5 to frame 295. This will cause the camera interest to start moving before the movement of the camera position, thereby keeping the camera interest always a little in front of the camera position.

- Play the animation or drag the frame counter o see the camera animaion you just made. Notice that the camera looks down the path as we intended – until the last few frames. This is because the camera position begins to catch up to the camera interest at frame 295. To fix this do the following.

- Use ***>Camera >Select Camera*** to deselect the camera interest and select the camera position. Use ***>FcrvSelect >Camera >Position >Translation*** (middle-left menu bar). The curve you see represents the percentage of the camera's position along the path at each frame.

- In the Function Curve window, click **EDITKEY** (top bar) and use the left mouse button to drag keyframe 300 (the center blue dot at the far right of the curve) down just a little. This will make the camera position travel a little less than 100% of the distance of the path. Drag the frame counter or play the animation to see the result. Close the Function Curve window by clicking on Fcurve with the middle mouse button.

- Save a new version of your scene under the same name

with ***>Save >Scene.***

Edit the shape of the path

- Once an object or camera is animated on a path, you can change the shape of the path and the object will automatically move down the new path. Drag the frame counter through your animation. If you need to change the shape of the path, select the path and make sure both its points and lines are visible with ***>Show >Points, >Show >Lines***. Then hold down the **m** key and drag any of the curve's points.

- Go to Matter and use ***>Preview >All*** to see what your animation looks like at various frames. Continue to edit the path curve until you have something you like.

- Save a new version of your scene under the same name with ***>Save >Scene.***

Render and view a flipbook

- Go to the **Matter** module and use ***>Render*** to create test frames of your animation. Set the frame count to 1 to 300 by 5, and the resolution to 320 x 240. Set the filename to *path*. Click **Render Sequence**.

- When the frames are finished rendering, go to the **Tools** module and use ***>Flipbook*** to view the rendered playback. Set Do frames 1 to 300 by 5 and Number of Frames per second to 6.

PART 4

Lecture 4

Hierarchies

Inverse Kinematics

Rotational Limits

Rigid Surfaces

Flexible Surfaces

Contraints

Hierarchies

A single model can be manipulated easily as a unit: if you want to move a cube, you pick the entire cube and just move it; if you need to make a vase larger, you pick the entire vase and scale it.

Complex objects often have many parts, however, each one of which might need to be transformed or animated individually. Animating a car, for example, will probably involve translating the whole car down the road. At the same time, you might need to rotate its wheels, and perhaps later open its door.

This sort of model requires a structure to allow you to select and then operate on specific elements of the model. The technique that provides this structure is **hierarchical modeling.**

The first stage in building a hierarchical model is to make sure that any elements which will need to be moved independently are in fact modeled as separate, independent surfaces. The door of the car, for example, must be a distinct object, not just an area of one long, continuous side panel.

The next step is to define which objects affect the transformations of which other objects. For example, the transformations of the car should affect the transformations of the wheels, in the sense that if you translate the car down the road, you want the wheels of the car to follow. Transformations of the wheels, however, should not affect the whole car: if I rotate a wheel, I do not want the whole car to rotate. Setting up these transformation relationships between each element of a model is what is meant by defining the hierarchical structure of the model. The whole structure is thought of as a **hierarchical tree**. (Figure 1-66) Each element of the model is a **node** in the structure. If one node affects the transformations of another node, the first node is considered a **parent** and the second is a **child** node.

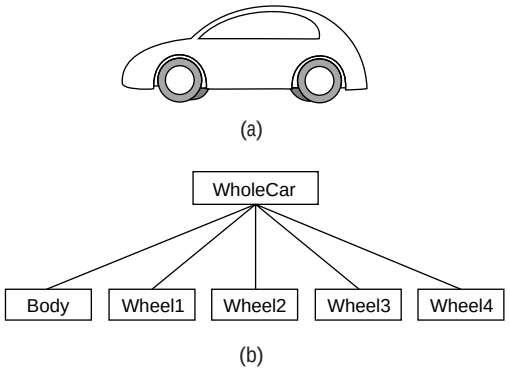


Figure 1-66. A hierarchical model and a schematic representation of that model.

In the example of our car, each wheel is a child of the whole car, and the whole car is a parent of each of the wheels. The transformations of the parent (*WholeCar*) transfer, or propagate, down the hierarchy to each of the children (*Wheels*). Transformations of the children, however, do not propagate upwards to the parent.

Inverse Kinematics

In describing hierarchical models, we saw that the transformations of a parent node propagate downwards to affect all of its children nodes. For example, if I translate the parent car down the road, all its children wheels also translate down the road. This kind of hierarchy exhibits what is known as **forward kinematics**. The word "kinematics" refers to the calculation of motion. The word "forward" refers to the fact that the transformations travel in a forward direction down the hierarchical tree, from parent to child.

There are some models, however, which are easier to animate in a different fashion. A human arm is a good example. For an arm, the top node in our hierarchical structure would be the upper arm. The middle node would be the forearm. And the bottom node (if we simplify the situation by eliminating fingers) would be the hand. That is, the upper arm is the parent of the forearm, which in turn is the parent of the hand. This gives us the structure we want: if we move the upper arm, all the rest of the arm (that is, the forearm and the hand) go with it; if we rotate the forearm, the hand moves with it, but the upper arm is unaffected. (Figure 3-27, a)

Making the arm reach up to scratch a character's head, however, is not very easy. In order to pose this forward kinetic model, we would have to first rotate the upper arm, then rotate the forearm, then rotate the hand. This process would have to be repeated, perhaps many times, refining the rotations of the joints each time, until we finally got the hand to touch the head.

To scratch our head in real life, we do not begin by thinking about rotating the upper arm. Instead, we think (if we become conscious of this movement at all) of our hand – we simply move our hand to our head. The movement of our hand forces our elbow to bend and our shoulder to rotate. This is an example of **inverse kinematics**. In inverse kinematics, or **IK**, the rotation calculations for each joint of the arm are calculated up the hierarchical tree, in the inverse direction, from child to parent – from the hand to the forearm to the upper arm. (Figure 3-27, b)

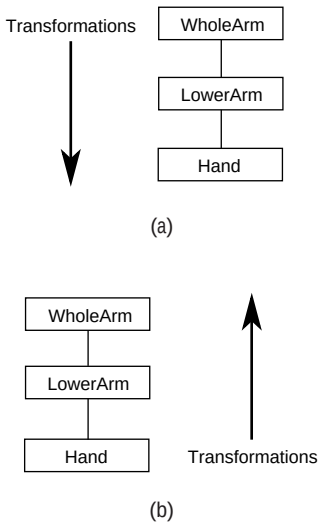


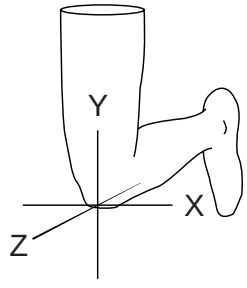
Figure 3-27. In standard hierarchical animation, transformations are calculated down the hierarchical tree. In inverse kinematics, the transformations are calculated upward from the bottom-most joint.

Most animal figures that have skeletons – including the human animal – are best animated with inverse kinematics. In fact, the inverse kinematics structure is often referred to as an inverse kinematics **skeleton**, with each segment of the skeleton being sometimes called a **bone**. The bottom-most point of the bottom-most bone (which corresponds to the tip of our hand in our example) is called the **effector**, since moving it effects all the other nodes. By carefully defining a good inverse kinematic skeleton, you can make the job of animating human characters and other creatures much easier.

Rotational Limits

In setting up a complex hierarchical model that will entail a complex series of transformations it is often helpful to limit the transformations a given element may have. The wheels of a car, for example, should be able to rotate a full 360 degrees around the axis of their axles. The rear wheels should have no other possible rotations. The front wheels, however, need an additional rotation in order to be able to "turn" the car. In addition to the rotation around their axles, they should also be able to rotate about +/- 45 degrees in the direction of the car's turning. These are all **rotational limits**, and each is defined in terms of a minimum and maximum rotation around each axis of the object.

Rotational limits become especially useful with inverse kinematic skeletons used for human and animal motion. This is because the joints of our bodies have natural, physiological rotational limits. For example, a human knee can rotate only about one axis (hopefully!) This is the axis about which it rotates when we bend our leg. If this axis is the X axis, then knee rotations about X are limited to about +130 degrees (bending the leg back) and 0 (leg straight, knee unbent). Rotations about the other two axes, Y and Z, are limited to zero – that is, no rotation at all (Figure 3-43).



(a)

	MIN	MAX
x	0	130
y	0	0
z	0	0

(b)

Figure 3-43. The joints of animal bodies have naturally occurring rotational limits.

Rigid Surfaces

Some models can be animated very effectively with surfaces which do not bend or deform. The doors of both the car and van we described earlier are examples. These need to either rotate or translate, but they do not need to bend or squish or squash as they open (unless we are making a very cartoony animation). Another example would be a robot. As a robot moves about, his body segments need to rotate and move, but they typically do not need to change shape. These are all examples of **rigid geometry** – surfaces which transform (move, rotation, scale), but which do not change shape. Rigid geometry can be very effective both for simple models and for more complex hierarchical models like our robot.

Rigid geometry can also be applied to an inverse kinematic skeleton. Most software packages require you to first design the inverse kinematic skeleton. Each rigid geometry part is then made a hierarchical child of the appropriate joint. (Figure 3-35) In a robot arm model, for example, an ellipsoid might be made a child of the forearm bone/node. Since the ellipsoid is now a child of the forearm node, any transformations of the forearm will be passed on to the ellipsoid. In other words, the ellipsoid model will go wherever the forearm bone goes – which is exactly what we want.

Flexible Surfaces

Some models, however, are most effective if the surfaces of the model bend and change shape as the model animates. A snake is a prime example. The muscles and skin of humans are another example. The leaves of a plant as they bend in the breeze are another example. All these are examples of **flexible surfaces**.

There are a great many techniques for creating and animating flexible surfaces in computer graphics. One which is particularly powerful allows the surfaces of an inverse kinematic

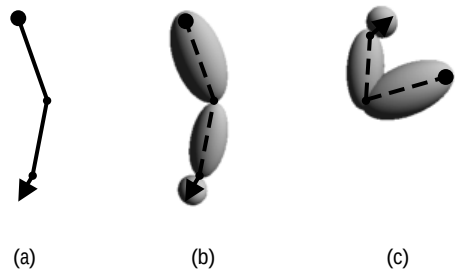


Figure 3-35. Individual models can be associated with each limb of an inverse kinematic skeleton. When the skeleton moves, the models move with it.

model to bend smoothly as the inverse kinematic skeleton bends. Surfaces which are animated with this technique are often called **envelopes.** (Figure 3-38) To use this technique, you first define the inverse kinematic skeleton – for example, an arm. Then you create a model consisting of a single continuous surface on top of the skeleton. The envelope technique then automatically calculates how this model should bend as the bones of the skeleton rotate. In actual practice, of course, the process is much more complicated than this, with the animator controlling a great many variables to determine

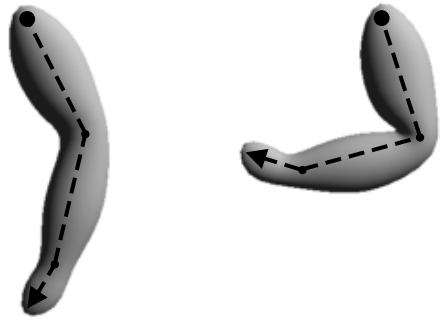


Figure 3-38. As the inverse kinematic skeleton deforms, the envelope associated with it also automatically deforms.

exactly how the envelope bends. But the basic idea is simple: by moving the effector of the skeleton, and thereby changing the orientation of the skeleton's bones, you automatically deform the flexible envelope. This mimics what happens when you move your hand to your forehead to scratch your head, and the skin and muscles of your arm automatically bend and stretch and contract.

Constraints

A technique which can be very useful in animating 3D objects is constraints. A constraint forces, or constrains, the motion of one object to mimic that of another. There are several kinds of constraint. Forcing your model to always point toward a constraining object is a **direction constraint**. Forcing your model to always rotate exactly as a constraining object is a **rotation constraint**. Forcing your model to always be located in exactly the same place as a constraining object is a **position constraint.**

In working with an inverse kinematic skeleton, for example, you might create a shoe model and use that as a position constraint to control the position of a leg's effector (Figure 3-47). In this way, if you want to move the character's foot or bend its leg, you can simply select the shoe and move it.

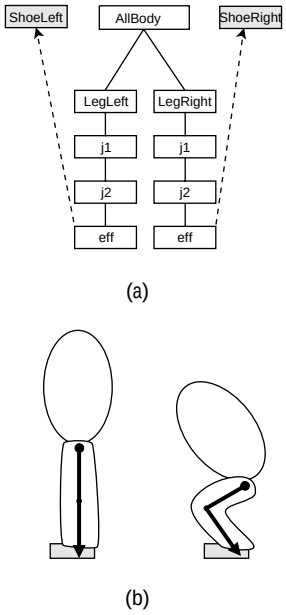


Figure 3-47. Constraining the position of an effector to another object can facilitate complex character animation.

Using a constraint with inverse kinematics can facilitate certain types of animation greatly. By keeping the constraints outside the normal hierarchy of the character, as in Figure 3-47a, you are able to move the constraints (and therefore the feet) completely independently of any motion that might be on the parent node of the whole hierarchy.

On the other hand, if you were to make a new node that was parent to both the old parent (*Allbody*) and also to the constraints (*ShoeLeft and ShoeRight*), you would now be able either to control everything at once by operating on this new node, or to control the feet independently by operating on the *Shoe* constraints.

Exercise 4

In this exercise we will make a simple inverse kinematic skeleton, apply some geometry to the skeleton to represent a simplified human figure, define a hierarchy and constraints to permit us to animate this figure, and then make a simple animation of the figure jumping up into the air.

If you prefer to skip the modeling portion of this exercise, you may simple use ***>Get >Scene*** to retrieve the scene *Exercise4-Eggmodel*, which is a working model as described in PART A, but without any animation. If you retrieve this file, you may proceed immediately to PART B.

PART A: MODEL A CHARACTER

1. Create the inverse kinematic skeletons

Create two leg skeletons

- Click on **Actor** (top bar) to go to the Actor module of Softimage. First we will make two legs of a figure. The figure will be facing us in the Front window.
- Use ***>Skeleton >Draw 2D Chain*** (middle-left menu bar) to draw a two-jointed leg chain in the Right window. Give this leg a little bend at the knee (that is, toward the left of the screen) as you draw it. End your chain by clicking the right mouse button.
- In the Front window, use ***>Trans X*** to translate this leg chain over a bit to the right.
- Use ***>Duplicate >Immediate*** to create a copy of this leg. Use ***>Trans X*** again to move this new leg a bit to the left in the Front window.
- In the Top window, click on the word Top and drag down to the word **Schematic** to display a schematic diagram of the scene information. You should see one hierarchical structure for each of the leg chains. Use the **z** key to zoom out of this window if necessary.

Test the movement of each skeleton

- Use ***>Select >Clear*** and then ***>Select >Toggle Mode*** to select one of the effectors. This is the little crosshair at the bottom of each leg. You can also select this from the Schematic window. Use ***>Trans*** to move this effector around. The leg skeleton should bend.
- Use ***>History >Undo*** (or just the **u** key and a click in the window) to undo this translation.

- Save your work with ***>Save >Save Scene***. Make the Prefix *Exercise4* and the Scene Name Eggman (or, Egglady?).

2. Create the geometry of the character

Make a torso

- Use ***>Get >Primitive >Sphere*** to create a sphere for the body of our eggperson. Make it a B-spline Patch.
- With the sphere still selected, click ***>Info >Selection*** (upper-left menu bar). Change the **Name** to *torso*. Notice that the name changes in the Schematic window.
- With the *torso* sphere still selected, use ***>Scale*** to make it the size of an egg-shaped torso. Then use ***>Trans*** to position it just above the legs.
- With the *torso* sphere still selected, click **CTR** at the far bottom right of the screen, then use ***>Trans Y*** to pull the sphere's center, represented as a miniature axis, down to the bottom of the sphere. When you have done this, click **OBJ** (far bottom right) to go back to Object mode.
- Save a new version of your scene, keeping the same names, with ***>Save >Save Scene***.

Make two feet

- Now use ***>Get >Primitive >Cube***, with Length = 1.0, to create a box for a foot. Use ***>Info >Selection*** to change the name of this object to *footL*.
- Use ***>Scale*** to make the cube longer and foot-shaped. Use ***>Trans*** to position it as the foot of the left leg.
- With the *footL* box still selected, click on **CTR** (bottom-right of screen). Use ***>Trans*** to move the center of the foot to the ankle position. Click on **OBJ** to go back to object mode.
- Use ***>Duplicate >Immediate*** to make a copy of this foot. Use ***>Info >Selection*** to name it *footR*. Use ***>Trans X*** to translate it along the X axis into position under the right leg.

- Save a new version of your scene, keeping the same names, with ***>Save >Save Scene***.

Make two legs

- Now we will make a flexible surface, or envelope, for each of the legs. Use ***>Select >Clear*** and ***Select >Toggle Mode*** to select one of the leg skeletons. The best way to do this is to select the top node of the leg hierarchy in the Schematic window.
- When this is selected, click ***>Skin >Automatic Envelope*** (lower-left menu bar). Then click with the middle mouse button to specify a default circular cross-section for the envelope. Accept all the default settings for the envelope surface. Notice in the Schematic window that the new envelope node has been made a child of the leg's first joint.
- Repeat this same process for the other leg. First , select the leg, then click ***>Skin >Automatic Envelope*** and then the middle mouse button to make a default, sausage-like envelope for the leg. Check your Schematic window to make sure both leg envelopes are correctly grouped within each leg skeleton
- Save a new version of your scene, keeping the same names, with ***>Save >Save Scene.***

3. Define more of the figure's hierarchy

Make the legs children of *torso*

- Go to the **Model** module of Softimage.
- Use ***>Select >Clear*** to make sure nothing is selected. Then use ***>Select >Toggle Mode*** to select the *torso* sphere.
- With *torso* selected, click ***>Parent*** (lower-right menu bar). In order to select a child of this parent, use the **left mouse** button and click on the top node of one of the leg chains in the Schematic window.
- The Schematic diagram will change, showing the *torso* sphere as the parent of this leg.

- With the *torso* sphere still selected, use the left mouse button to click on the top node of the other leg. This leg skeleton becomes a child of *torso*.
- End Parent mode by clicking the **right button**.
- Save a new version of your scene, keeping the same names, with ***>Save >Save Scene***.

Constrain the effectors to the feet

- The best way to control the feet of the figure will be to use constraints, rather than putting them inside the figure's hierarchy.
- Click **Actor** (top bar) to go back to the Actor module of Softimage.
- Use ***>Select >Clear*** and then ***>Select >Toggle Mode*** to select the effector of the left leg. It may be easiest to do this in the Schematic window.
- With this effector selected, use ***>Constraint >Position*** (middle-left menu bar) and click on the *footL* box.
- In the top bar of the Schematic window, click on **Model Mode** and drag down to the word **Model**. A yellow line will appear between *footL* and the effector of the left leg to show that *footL* is constraining that effector.
- Repeat this process for the right leg. Select the effector of that leg, then use ***>Constraint >Position*** and click on the *footR* box.
- Test the constraints you just made. Select only the *footL* box and use ***>Trans*** to translate it. The left leg should bend just as if you were translating the effector itself. Use ***>History >Undo*** to move the *footL* box back to where it was.
- Save a new version of your scene, keeping the same names, with ***>Save >Save Scene.***

Complete the figure's hierarchy

- Finally, we need an extra node at the top of our hierarchy that will control everything – the torso, the legs and the

feet.

- Go back to the **Model** module.
- Use ***>Get >Primitive >Null*** to create a null node (that is, one with no geometry). With the null still selected, click on ***>Info >Selection*** and change the Name of this node to *All*.
- With the *All* null still selected, click ***>Parent***. Using the left mouse button, click on the *torso* sphere. This will make the whole torso stucture a child of *All*. (Remember the legs are already children of *torso*).
- Still in parenting mode, click with the left mouse button on the *footL* box and then on the *footR* box, to make them children of *All*.
- Look at the Schematic window. *All* should be the parent of both of the feet and also of the torso structure. *Torso*, in turn, is the parent of each of the legs. Within each leg, you should see the envelope for each leg. Finally, the effector of each leg should be constrained to its foot model.
- Save a new version of your scene with ***>Save >Save Scene***.

Test the complete hierarchy

- Before beginning to animate, test the structure you made.
- Use ***>Select >Clear*** to un-select everything. Use ***>Select >Toggle Mode*** and click with the **middle mouse** button on *All* in the Schematic window – that is, the very topmost node of the hierarchy.This will select all elements of the hierarchy. With all elements selected, use ***>Trans*** to move the character around. All components of the character should move together. Use ***>History >Undo*** to undo your translation of the character.
- Unselect everything again and use ***>Select >Toggle Mode*** and the middle mouse button to select the *torso* sphere in the Schematic window. This should select the body and legs, but not the feet or *All*. Use ***>Trans*** to move the figure up and down. The body and legs should move as a unit, but the feet should remain where they are. This is because they are outside of the torso hierarchy. Use ***>History >Undo*** to undo your translation of the character.

PART B: ANIMATE THE CHARACTER

1. Set up

- Now we will make the figure crouch and then jump up into the air. At each keyframe, we will save transformations for each of the components we named in Part A. That is, we will save a keyframe for *All*, for *torso*, for *footL* and for *footR*.
- Go to the **Motion** module. Make sure your ending frame is 100.

2. Define an initial standing pose

- Frame 1 will have the figure standing upright. Move the frame counter slider to frame 1.
- Select the torso with the middle mouse button. This will select the *torso* sphere and both of the legs. With these selected, use ***>Trans*** to position the figure so that it is upright and at rest. That is, move the torso sphere so that it is over the legs and the legs are straight. When you have a pose that looks good, use ***>Save Key >Object >Explicit Translation > All.***
- Select *footL*. If necessary, translate it into a good standing position – that is, directly underneath the torso. Use ***>Save Key >Object >Explicit Translation > All.*** Do the same thing for *footR*.
- Finally, select *All* with the middle mouse button. This will select the entire hierarchy. Do not move this at all, but use ***>Save Key >Object >Explicit Translation > All*** to save its position.
- Save your scene with ***>Save >Save Scene***. Use *Exercise4* as the Prefix and *EggmanAnim* as the Scene Name.

2. Define a crouched pose

- Advance your frame counter to frame 30. Use ***>Select >Clear*** to unselect everything, then use ***>Select >Toggle Mode*** and click with the middle mouse button on the torso sphere, to select the torso sphere and legs. Translate this

straight down so the figure is in a crouching position. When it looks good, use ***>Save Key >Object >Explicit Translation >All***.

- Unselect everything again, and select *All* with the middle mouse button to select the entire figure. It is not necessary to translate *All*. Even so, we should define a keyframe for it to make sure it remains unchanged. With *All* selected, use ***>Save Key >Object >Explicit Translation >All.***

- Do the same for each of the feet. Even though they should not move for this keyframe, select each one and save a keyframe for it with ***>Save Key >Object >Explicit Translation >All.*** to make it stay where it is. Do this for both feet.

- Drag the frame counter triangle back and forth. You should see the character move from a standing pose at frame 1 to a crouched pose at frame 30.

- Save a new version of your scene with ***>Save >Save Scene,*** using the same name, *Exercise4- EggmanAnim.*

3. Define a rising pose

- The next pose we want is when the character has started to jump but has not yet left the ground.

- This pose will be very similar to the standing pose at frame 1, except that the legs will be slightly bent. The easiest way to make this pose is therefore to make a variation of frame 1.

- Move the frame counter to frame 1 to see the character standing upright. Use the **right mouse** button to slide the frame counter to frame 40. This will leave the figure in its frame 1 pose at this new frame.

- Unselect everything with ***>Select >Clear*** and then select *torso* with the middle mouse button to select the torso and the legs which are under it. Use ***>Trans Y*** to move this down just a little so that the legs bend slightly. Use ***>Save Key >Object >Explicit Translation >All*** to save this position of *torso*.

- The other components do not need to be translated, but they do need to have keyframes saved for them. Unselect

everything and select *All* and use ***>Save Key >Object >Explicit Translation >All*** to save its translation. Unselect everything again and select *footL* and use ***>Save Key >Object >Explicit Translation >All*** to save its translation. Do the same for *footR.*

- Test your animation so far by dragging the frame counter back and forth. The figure should be standing straight up at frame 1, crouched at frame 30, and standing but with legs bent at frame 40.
- Save a new version of your scene under the same name.

4. Define a mid-air pose

- The next pose will be when the figure has jumped into the air. Advance your frame counter to frame 50.
- Begin by selecting *All* with the middle mouse button. This will select the entire hierarchy. Use ***>Trans*** to translate this up into the air, then use ***>Save Key >Object >Explicit Translation >All*** to save this translation at keyframe 50.
- Unselect everthing and select *torso* with the middle mouse button. This does not need a new translation, but save a keyframe for it with ***>Save Key >Object >Explicit Translation >All.***
- Unselect everything and select one of the feet. Translate it so that it looks like the figure has jumped. For example, translate the foot up into the air and away from the body. Use ***>Save Key >Object >Explicit Translation >All*** to save this translation for that foot.
- Do the same for the other foot.
- View your animation by dragging the frame counter.
- Save your scene under the same name with ***>Save >Scene.***

5. Add some rotations to the torso

- The figure is now jumping, but its back is very rigid. To make it more lifelike, we can add some rotations. Unselect everything and select *torso* with the middle mouse button.

- Go to frame 1. The *torso* will remain as is for this frame, but we must save a rotation keyframe for it anyhow. With torso selected, use ***>SaveKey >Object >Rotation >All.***

- Keep *torso* selected. Advance to frame 30. Use ***>Rot X*** to rotate the torso forward a bit into a more natural crouching pose. Use ***>SaveKey >Object >Rotation >All*** to save that rotation.

- Advance to frame 40. Again, rotate the torso into a natural pose and save a rotation keyframe for it with ***>SaveKey >Object >Rotation >All.***

- Advance to frame 50, rotate the torso, and save its rotation.

- Look at your animation by using the Play triangle in the far lower right of the screen. Don't forget to go into ***>Play Control*** and set **Frame Step** to 0 to force Softimage to play the animation at 30 frames per second.

- Save a backup version of your scene with ***>Save >Scene.***

6. Render and view a flipbook

- When you have an animation you like, go to the **Matter** module and define some materials for your surfaces. If you don't like the default light, you can also define some lights for your scene. Advance your frame counter and use ***>Preview*** to see what your rendering looks like at several different frames of the animation.

- Use ***>Render*** to render small test frames of the animation. Set the **Resolution** to 320 x 240 as we did in the earlier exercises. Set **Start** = 1, **End** = 100. However, instead of setting **Step** to 5, as we did in the other exercises, set **Step** = 2. This will give us a smoother, more accurate view of the character's motion.

- After your frames are rendered, go to **Tools** and use ***>Flipbook*** to view them. Don't forget to set the **Do frames**, **to** and **step** numbers to the same numbers you used when rendering – that is, 1, 100, 2. Since step = 2, set **Number of frames per second** = 15 to give you the correct playback speed (2 * 15 = 30).

How's that for a day's work?

Bibliography

Muybridge, Eadweard, *The Human Figure in Motion*. New York, Dover, 1955.

Muybridge, Eadweard, *Animals in Motion*. New York, Dover, 1957.

O'Rourke, Michael, *Principles of Three-Dimensional Computer Animation*. New York, W.W. Norton,1998, Revised Edition.

Softimage Inc., *Softimage Documentation and Reference Manuals*. 1996-1998.

Thomas, F. & Johnson, O., *Disney Animation: The Illusion of Life*. New York, Abbeville, 1984.