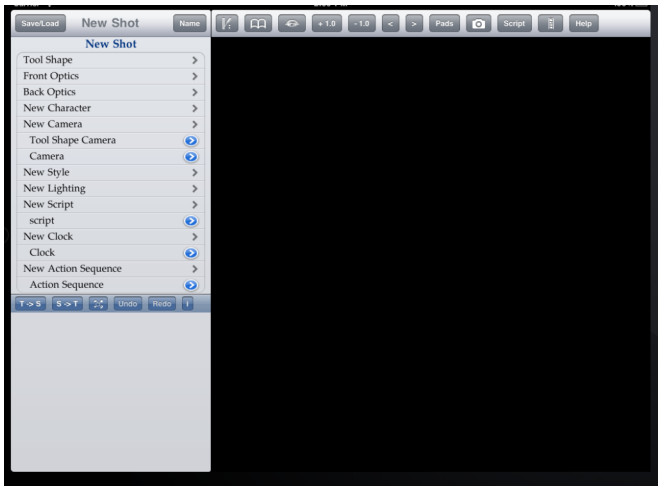
IgorIpad Manual



IgorIpad is a compact version of IgorAnimation's animation and image processing program that allows animations to be created and images to be transformed on your iPad.

IgorAnimation works differently from conventional animation. Rather than creating fixed models, IgorAnimation creates highly flexible, easily transformed models. IgorAnimation models consist of mathematical specifications of how the models are to be drawn. These instructions can be as simple as the radii of an ellipse or as complex as you care to make them. IgorAnimation opticals are also specified by mathematical instructions which can be used to create colorings on their own or modify the colors of precreated images. Opticals can be used to color models or to be drawn on their own. This allows IgorAnimation to be used both as an animation program and as an image manipulation program.

The models and opticals created in IgorAnimation take less memory and less processing power than conventional CGI animations and are capable of arbitrary levels of sophistication.

IgorIpad has two main areas: the view, which is the blank area under the toolbar, and the structure, which is the table view to the left of the view in landscape mode (in portrait mode the structure appears in the Structure popup). The structure contains the objects that will be drawn in the view as well as other objects (such as cameras) that determine how things will be drawn.

IgorAnimation is versatile, allowing for a great variety of objects to be created and animated. It's pretty easy to use once you get used to it, which is what this manual is for. This manual is organized in sections with tutorials for how to use the various parts of the program. Each section teaches the use of one or more of the popup buttons along the top. The sections more or less follow the popups from left to right. One of these popups, the appearance controls (the eye), affects many aspects of viewing the models, so it will be referred to control by control in the following sections. Instructions for each popup can be found by tapping the

Help button, then tapping the button in the help toolbar that looks the same as the button in the main toolbar.

1. Structure, Controls and Library: In this section you'll learn to build and modify models and opticals. You'll also learn how to find out the specifics of how each object in IgorAnimation works.

2. Pads: In this section you'll learn to control the appearance of your models with standard iPad touchscreen motions. You'll also learn to use these motions to puppet your models, recording their changes for animation.

3. Snapshots and Images: In this section you'll learn to bring pictures into IgorAnimation and make single-frame pictures from IgorAnimation.

4. Scripts and Animation: In this section you'll learn to create instructions that tell your models and images how to move and change over time. Then we'll take those instructions and make animations from them.

5. Advanced Topics. IgorAnimation has some sneaky tricks inside it that allow for farther-reaching usages. In this chapter we'll talk about some of them.

Section 1:
Structure, Controls, and Library

For most of what follows, it's probably best to hold the pad in landscape mode. This way it's easier to see the structure. Most of the rows in the structure represent the components that make up the whole of what one is looking at.

IgorAnimation objects are created by combining the instructions for simple-to-draw things in ways that create complex drawings. This tutorial centers on creation and manipulation of some simple-to-draw things that point toward more complex drawings.

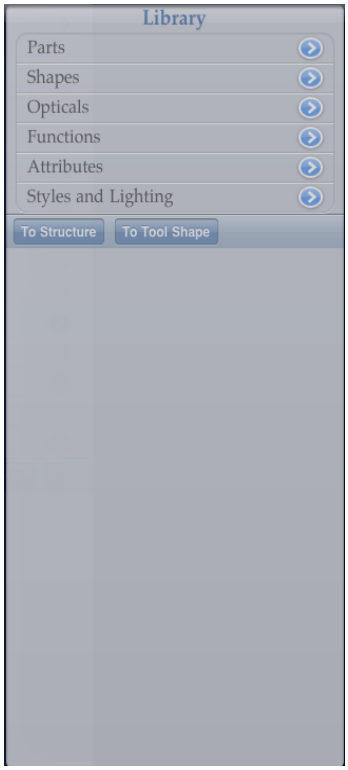
In order to create the complex out of the simple it's a good idea to be able to work on the simple objects first and then place them into the complex structure. IgorAnimation does this by having the ability to look at one single piece at a time, separate from the whole. In order to make this easy, the top row in the structure (the row labeled Tool Shape) is a spot to put a single object to work on before placing it into a more complex object. The view area will either show the tool shape or the entire structure except for the tool shape. When you start up IgorAnimation, the view is set up to show the tool shape. There's nothing to see at the moment because we have not yet chosen a tool shape.

Shapes

We'll start in a simple way and create an ellipse that we'll use as tool shape. We have an entire library of objects, basic and complex, to pull up as we need and place where we want in the structure. The contents of the library are classes of objects, so each time we take something from the library we are making a new object of that class.

Tap the icon on the toolbar above the view.

You'll see a popup that looks like this:



Tap the blue arrow button at the end of the row labeled Shapes. A new section labeled Shapes will appear in the library. Notice that some of the entries in this section are in black; these represent categories of objects, sections of the library, as it were; and others are in purple; these represent individual types of objects. To open a category, tap the blue button at the end of the row (as you already did with shapes). Opening a category creates a new section of the table with the contents of that section of the library.

Open the Basic Shapes category in the Shapes section and scroll down until you can see the row labeled Ellipse. Select this row, and tap the button in the toolbar at the bottom of the library labeled "To Tool Shape". An ellipse should appear in the view.

Take a look at the tool shape row in the structure. It's now labeled Tool Shape: Ellipse. Also note that the > at the end of the row has become a blue disclosure button. This means that the object in the row has its own components which can be changed in order to make it more interesting. Tap this button to open a new section, and a new section labeled Ellipse will appear in the structure.

The rows in this new section are the components of the ellipse, which represent various things that can be changed in order to determine how the ellipse will be drawn. Some of these components can change it so radically that it will not resemble an ellipse at all. We'll start with the simple components.

Look at the first three rows of the Ellipse section:

First Radius	30.0000
Second Radius	30.0000
Height	0.0000

Each of these has a number after it. Any component with a number is called a value component. It represents some number that has a specific role in determining how the object is drawn.

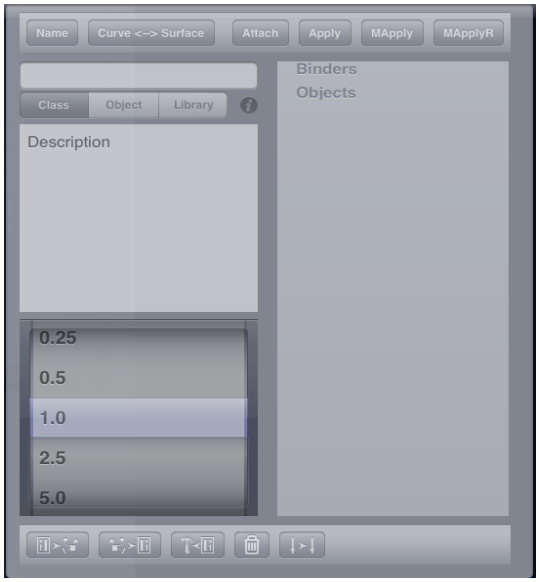
The first radius is the radius along the x axis, the second radius is the radius along the y axis, and the height is the z coordinate. These values can be changed in several different ways. The simplest uses these four buttons in the main toolbar:

These buttons are used to change the selected value component in the structure. Let's start with the first radius. Select the First Radius row, then tap the button labeled +1.0. The radius will go from 30.000 to 31.000 and the ellipse will redraw slightly wider. Click the +1.0 button again to keep raising it. To lower the value, tap the -1.0 button. Stop doing this before you get annoyed with the slow progress of change.

The amount by which these two buttons raise or lower a value component is called the increment. The increment can be changed using the < and > buttons. To raise the increment, click the >. To lower it click <. The + and - buttons change their labels to reflect the current increment. Click > until the + and - buttons say +25.0 and -25.0, then click the button now labeled +25.0. That's better.

But suppose you want to very carefully change a value by a tiny amount. Click the < button until the buttons say +0.01 and -0.01. Then change the radius. These values are too small to make much difference, but fine tuning like this will come to matter later.

Suppose you don't want to fiddle around like this and you simply want to set the value component to a specific number. Let's do that for the second radius. Select the second radius, then tap this button in the toolbar. A popup, called the Controls popup, will appear. It should look like this:



The text box on the top left should have in it the current value of the second radius. Tap in the box, and type in a new value when the keyboard pops up. When you dismiss the keyboard the second radius will change to whatever value you typed.

As long as we're here, let's take a look around. In the lower left of the Controls popup is a spinning picker. This is used to set the increment. Spin this to an increment you want. Notice that the + and - buttons in the main toolbar have changed to this increment.

This popup also contains a text area that you can use to get information about any object in the structure table. Select the tool shape (that is, the ellipse) in the structure, and go back to the Controls popup. Look at the control that has three segments :

. This control specifies what you are getting information about. Make sure that Class is selected. Tap the little i button. The box below will fill up with a full description of how ellipses work in IgorAnimation, including a full description of every row in the structure for that object.

Read through the description of the ellipse. It talks about three different ways the points of the ellipse are calculated: a simple one (which we've already seen) where the two radii and the height are constant values, a more complicated method in which those values can be calculated using functions, and an even more complicated one which we won't get into in this tutorial. To get to the second version we need to do two things: flip a switch and supply one or more functions. The switch is labeled Open? and is found in the ellipse listing in the structure. Flip the switch from Off to On. Notice that nothing whatsoever has changed in the view, but in the structure the ellipse is now called a Variable Ellipse. To actually change what we're seeing we need to supply functions to generate values for the radii.

A function in IgorAnimation is a means of calculating numbers from other numbers. For mathematical and programming pedants, we know that this is not a full or exact definition, but we don't need a fuller or more exact definition here.

If you look in the variable ellipse listing in the structure you'll see a row labeled First Radius Function. Select this row. Now, open the Library popup and tap the blue button in the Functions row of the first section. Where before we were looking at shapes (which generate points in space), we are now looking at functions (which generate numbers). Select the function called Sine, then tap the To Structure button at the bottom of the Library popup. Notice that the shape has changed (into what looks like a line but really isn't, as we will see) and that the First Radius Function row now has more words in it and its > has turned into a blue button.

So what did we just do? We took a row representing a component that can hold an object (the first radius function) and used the library to put an object in it. This kind of row is called an object component. Specifically it's an object component of the variable ellipse. Objects in IgorAnimation can have three kinds of components, all of which we've now seen:

1. Value Components: These have numbers assigned to them. Changing the numbers will change how the object is drawn. Each value component serves some specific purpose in the drawing of the object. For example, the first radius component specifies the radius of the ellipse along the x axis; and the height component specifies the z coordinate for each point.

2. Switch Components: These have switches in their rows. Turning the switch on or off will affect the behavior of the object.

3. Object Components: These have objects assigned to them. Each object component has a role in the way the object is drawn. Assigning different objects to an object component will produce very different kinds of drawings from the same base object. Note: Each object component will only accept certain kinds of objects. The first radius function can only have a function assigned to it; it would get no effect from a shape, for example.

Let's get back to the shape. Why does it look like a line? Can we do better than this? To understand and alter what's going on we need to open up the function that 's supplying the first radius and mess with it. Tap the blue button in the First Radius Function row.

A new section of the structure table has opened. This one contains the components of the sine function (which is labeled with the more general term Periodic Function).

The values the periodic function produces are being used to determine the first radius of the ellipse. But this radius is being recalculated for each point on the ellipse. In order to see what this periodic function is doing we'll play with two of its value components: amplitude and frequency. Amplitude represents the

maximum value a periodic function can produce (it will vary between -amplitude and amplitude). Amplitude starts out at 1.0, which means the first radius function will never be more than 1.0 or less than -1.0. That's why it looks like a line. It's actually just very thin. Use any of the means we've already covered to raise this value to around 30.0.

We've got a figure 8 on the screen. Why? Because the first radius is varying between -30.0 and 30.0. We can make the figure even more complicated by increasing the frequency, which tells you how many times the value of the function will vary between -amplitude and amplitude. Try raising the frequency from 1.0 to 5.0 using the +1.0 button, to see what different frequencies produce.

You may notice some angularity in the drawing. That's an effect of the resolution we're drawing at. We'll discuss raising and lower resolution later. We're keeping resolution at a medium level since it makes drawing faster, even though it gives us some artifacts. They'll all go away before we render anything.

So far we've changed only the first radius function. The second radius is still whatever constant value you left it at. Let's put a function in that as well. This time we'll do something unnecessarily complicated to introduce another feature.

First select the Second Radius Function in the Variable Ellipse section of the structure. Go back to the library. Go to the Exponential Function and tap the blue arrow. Nothing seems to have happened. Now go back to the Controls popup. The formerly empty table in the right half of this popup now has an entry labeled Exponential in the table section called Objects. This table is called the tray, and it's where you can keep any objects you might have uses for at some later time. Select the exponential in the tray and tap the (From Tray To Structure) button. This assigns the object selected in the tray to the selected object component in the structure.

Once again the shape has changed. We'll now change it some more by messing with the components of the exponential function we just assigned.

Tap the arrow in the Second Radius Function row. This will open up a section for the exponential function we assigned to the second radius. Raise the Power component from 1.0 to 6.0 using the +1.0 button, and watch as the shape changes. Notice that now what is changing is the y coordinate of each point, whereas the periodic function was changing the x coordinate.

It can be a little hard to see what the second radius function is doing because of what the first radius function is doing, so let's temporarily get rid of the first radius function. Select the first radius function and tap the button (Empty). The first radius function is now gone and you can see what the exponential is doing with the second radius.

Of course, we've lost the first radius function and we might want it back. No problem. Tap the Undo button in structure toolbar. This will undo the removal.

Undo, however, is not helpful if we want to remove the periodic function, do something else, and then put it back. This is what the tray is for, to hold things for later use.

Assuming you've undone the removal, select the first radius function and tap the (Structure To Tray) button in the Controls popup. The periodic function has now been added to the tray, which means you can remove it from the structure and put it back later if you want. You can even remove it and put it somewhere else if you care to. For example, select the second radius function, select the periodic in the tray and tap the From Tray button. The exponential has been replaced as second radius function. Now select the first radius function and assign the exponential to it from the tray. The same functions are now serving opposite roles in the variable ellipse than they were a minute ago.

So far, we've been mucking about with a curve, a one-dimensional object. But to create the appearance of 3D objects we actually use surfaces, which are two-dimensional. Is that clear?

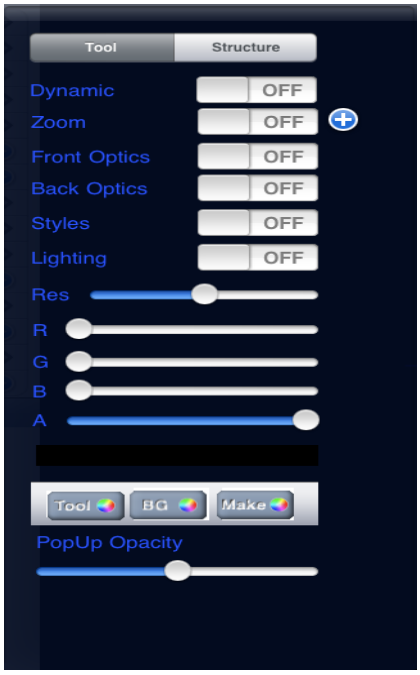
No. Okay.

Here's a bit of mathematical terminology that will be confusing compared to computer animation terminology. In animation, 2D means drawn on a plane. 3D means drawn in space. In mathematics, 1D means it's basically a bent, twisted, and possibly broken line segment. 2D means it's basically a bent, twisted, and possibly filled-in square. This is a very sloppy description that isn't fully accurate, but it's close enough for this purpose. Mathematics distinguishes the dimension of the object from the dimension of the space the object is put in.

In any case, to get objects that look like solid objects in space, we put 2D objects into 3D space.

Let's start with an ellipsoid, which can be found in the Shapes -> Basic Shapes section of the library. Select Ellipsoid and tap To Tool Shape.

Yes, it does look a little angular. That's resolution again. To see it at higher resolution, we need to go to the popup that controls how things look, the View Controls popup. Tap the button and this pops up:



This popup is used mostly to control aspects of the appearance of the view, but it has a couple of other uses which we'll get to later. The slider labeled Res controls the resolution. Slide it all the way to the right to see the ellipsoid much smoother. You'll notice that it takes longer for the ellipsoid to draw at high res. In general it's a good idea to work at low or medium res and use high res for taking snapshots and renderings.

Slide the slider back to the middle.

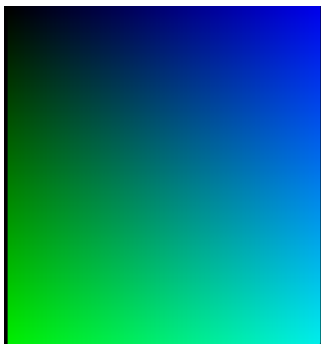
You can muck with the ellipsoid just as you did the ellipse by opening it in the structure and varying the values of its radii. But before you consider flicking the Open switch and making a variable ellipsoid of it, we need to take a brief look under the mathematical hood to see how the points of these shapes are being calculated.

Up to now we've been fairly cavalier about the numbers that generate the points of shapes, and the values of functions. Now it's time to take a closer look at these numbers that are fed in, the parameters of the functions and shapes.

Fortunately, this is a graphics program. There's no need to look at numbers as numbers when we can see the numbers as colors and get a better sense of how they work. To do that we need to go back to the View controls. The control we're after is the switch labeled Dynamic. Turn it on, and watch the color change on the ellipsoid.

So what's with the blue and green colors?

If you imagine the surface of the sphere as a bent and distorted square, then each point on the square becomes a point on the sphere. If we color the square like this:



then you can see how the colors on the square correspond to those on the sphere. The top left corner becomes the north pole. The colors grow progressively bluer as one goes around the sphere and progressively greener as one goes down. The south pole (which we'll look at in a minute) is pure cyan.

So what does this have to do with mathematical calculations?

The points on any surface in IgorAnimation are calculated from pairs of numbers, each between 0.0 and 1.0. The first number in the pair is called the first parameter. The second number is called the second parameter. The blue and green square is a visualization of this. The upper left corresponds to the parameters (0.0, 0.0); the upper right is (1.0, 0.0); the lower left is (1.0, 0.0); and the lower right is (1.0, 1.0). The more blue, the greater the first parameter, the more green, the greater the second parameter.

The ellipsoid draws its points from the parameters by calculating the x,y, and z coordinates as follows:

x = first radius * cos(2π * first parameter) * sin(π * second parameter)
y = second radius * sin(2π * first parameter) * sin(π * second parameter)
z = third radius * cos(π * second parameter)

This is a pretty standard formula. Every IgorAnimation shape has some such underlying formula that calculates the x,y,z coordinates from the parameters. In most cases, understanding the formulas is not necessary in order to use the shape. You can see the shape as it is drawn and vary its components to get a desired effect

If you need to look more closely at how something is being drawn, use dynamic coloring.

What happens when we open the ellipsoid and make it a variable ellipsoid? The first, second, and third radius are no longer constant values but are themselves calculated by functions that use these same parameter values. Exactly how functions use the parameters will be explored a little later.

For now let's shift our point of view and talk about shifting our point of view. Right now we're looking down at the ellipsoid from the north pole. To start looking around it we need to move the camera. There are two ways to do this. We'll get to the second one later in the section on pads. For now, we'll move it the same way we changed other things, by using the +- buttons on value components.

Go to the structure entry labeled Tool Shape Camera and click its blue button to open. Then tap the blue button of the entry labeled Angles (...). This gives you three value components to play with, which represent the position of the camera in space.

The camera is looking at a point (the camera's focal point) which right now is (0.0,0.0,0.0), which by coincidence (ha!) is the center of the ellipsoid in the screen. The camera is at a distance of 100.0 from that point. If you imagine a sphere with a radius of 100.0 centered on the origin, the camera can be at any position on that sphere. We specify that position by two angles. The second angle is how far down the sphere it is positioned (0.0 is the north pole, 90.0 is the equator, 180.0 is the south pole). The first angle is how far around the sphere we are (from 0.0 to 360.00). If these angles sound like they correspond to the way the first and second parameters are used in drawing the ellipsoid, they do. The third angle is called the spin angle and it rotates the camera so you can turn things upside down if you want to.

Let's start by going from the north to the south pole. Set the increment to 10.0. Then select the second angle in the structure and start clicking on the +10.0 button until the second angle is 180.0. Notice the color difference between the points near the north and south poles (the north pole has no green, the south is fully green and blue, hence it's cyan). In terms of the parameter square we've migrated from the top of the square to the bottom.

Make the second angle 90.0. We're now looking at the equator.

Now select the first angle, make it 0.0 and click through by +10.0s until you get to 360.0. You can see the way the sphere's blueness grows from 0.0 to 1.0 as you progress around.

But how does this actually look on the sphere without the dynamic coloring? Go back to the View controls, switch off the dynamic coloring and move the camera around once again.

Now let's flick the Open switch on the ellipsoid and start seeing what can be done with a variable ellipsoid.

Let's start by assigning the periodic function to the first radius function. Set the amplitude to about 60. Now, go back to the tool shape camera and move around the shape (use dynamic color if it helps you). See how the first radius varies as you move around the variable ellipsoid. What's happening is that the periodic

function is getting its values from the first parameter. This is what all functions do, unless you tell them otherwise. You can change what parameter is providing the values by using a special function called Parameter and assigning it as the suboperation of the periodic function. Select the variable ellipsoid in the structure and open it with the blue button. Tap the blue button next to the First Radius Function: Periodic Function, then select suboperation in the periodic function section of the structure. The suboperation is an object component that can hold any function. The value the suboperation produces is used as the parameter value for the periodic function.

Go to the library, open Functions -> Parameter Functions, select Parameter and tap To Structure. Parameter has been assigned to the suboperation, but nothing appears to happen to the shape. Tap the blue button next to suboperation:Parameter. You'll see a single new line labeled Parameter with the number 1 next to it. This means the parameter function is giving the first parameter value to the periodic of which it is the suboperation. Change that 1 to a 2. The shape changes dramatically as the first radius is now being determined by the second parameter.

Play around with the variable ellipsoid and various functions for its radius functions. When you've had enough, try some of the other shapes in Basic Shapes. If you need information about the shape or any of the functions, don't forget the i button in the controls.

Opticals

An IgorAnimation object has (as we will see) two essential components for its appearance: shape, which gives it shape, and optical, which colors it. Just as shapes have mathematically determined points, opticals have mathematically determined colors. Eventually we'll be making wild twisting shapes and distortions of pictures, but we'll start out very simply.

First, before you do anything else, go to the View controls and set the resolution all the way to the left (the lowest res possible). Opticals take up the entire view area and can take quite a while to draw at high or even medium resolution.

We'll start with the simplest and most boring of opticals, a color. A color is the same color everywhere. In the library open Opticals, select Color and make it the tool shape. Yes, the tool shape does not have to be a shape. The screen should turn white. In the structure, open up the tool shape to see what we've got.

Color has four value components: Red, Green, Blue, and Transparency. Together these make a standard RGBA color. The values of these components can be any numbers between 0.0 and 1.0. If you're not familiar with RGB colors, play around with the values to see what you get.

There's a faster way to generate a particular color you might want. Open the View Controls popup. Notice the RGBA sliders near the bottom. To create a color, set these to whatever you want. The color itself appears in a box below the

sliders. When you have a color you like, tap the

button. This places a copy of the color you just made into the tray. The other two buttons next to the Make color button are used to set other colors for other purposes. The

button sets the color that tool shapes are drawn with, and the

sets the background color for the view.

When you've had enough playing around with colors we'll get into some more IgorAnimationesque opticals. We'll start with an optical that is made from functions, the coloration. Open Opticals in the library, select Coloration and make it the tool shape. You will have a completely black view. Open the coloration in the structure. You will see four object components: Red Function, Green Function, Blue Function and Transparency Function.

Coloration uses function values to generate red, green, blue, and transparency values to create a color at each point. If there is no red function every color has 0.0 for a red value, similarly for green and blue. But if there is no transparency function, every color has transparency 1.0 (opaque).

Select the red function and assign the parameter function called Parameter to it (it's in Functions -> Parameter Functions). We now have an optical that grows from no red on the left side to fully red on the right. Open the Red Function : Parameter in the structure and change the parameter value from 1 to 2. Now the red is going from nothing on top to fully red on the bottom. This is how opticals treat the parameters. First parameter goes 0.0 to 1.0 from left to right, second parameter goes 0.0 to 1.0 from top to bottom, just as we shown before with the dynamic coloring.

Okay, but that's pretty dull. Let's see what we can do just by mucking with the parameters. Select the red function again and assign it Functions -> Parameters -> Parameter Sum. This looks just like the first parameter, but it isn't. This is a function that adds the parameter values. Open the Red Function -> Parameter Sum in the structure.

There are five switches, labeled Uses 1 to Uses 5, and five values, labeled Factor 1 to Factor 5. The switches determine which parameters are being used (don't worry about 3,4, and 5; they appear in some special objects, some of which we'll deal with later). Flick the Uses 2 switch to On. Now we have something that's adding the first and second parameter values to give us our red component. Notice that a fair amount of this area is fully red. That's because a coloration treats any value above 1.0 as 1.0, so that near the bottom right where both the first and second parameters are above 0.5, the total is above 1.0, hence totally red.

Now let's start messing with this. Parameter sum doesn't actually add the used parameters, it adds the factors times the used parameters. So what we really have here is factor 1 * parameter 1 + factor 2 * parameter 2. Try changing the first and second factors to see what you get. Notice what happens when one of those factors is negative. You end up with a completely black area where the sum is negative. That's because color values below 0.0 are treated as 0.0.

Set the factors to values you find amusing and we'll move on to green, blue, and transparency. We'll also explore the other parameter combinations. But rather than giving a bunch of boring instructions to go through slavishly, here's how to play around with this. Pick a parameter combination function from the Functions -> Parameter Functions section of the library. Before assigning it to the green, blue, or transparency function in the coloration, select it in the library, then go to the Controls popup. In the

segmented control, select Library. Then click the i button. This will give you information on the selected entry in the library. Decide if that looks interesting to you, then put in the structure and play around with its values.

You are, of course, free to put any other functions in as RGBA functions. Just remember that values below 0.0 are treated as 0.0 and values above 1.0 are treated as 1.0.

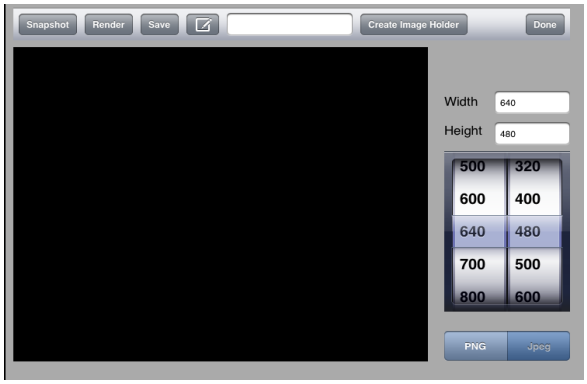
As you see you will be able to create any number of complicated visuals just with the coloration. But they tend to be kind of abstract. If you're more into realism and surrealism you need to be able to alter images that are taken from another source, like reality, say.

Image Manipulation

As seen above, colors are just batches of numbers, and numbers can be messed with by functions to get other numbers which can in turn be batched together to make new colors. Do this to the colors in a digital photograph and you get image manipulation.

IgorAnimation is very flexible in this because of how many functions and ways to play with numbers we put at your disposal. But first we need to get at an image. The following requires that you have some photos stored in your ipad. If you don't, please use iTunes to synch some pictures in.

All set. We're going to pull up a new view that's primarily used for taking screenshots and rendering images, but it is also usable for bringing images into IgorAnimation. Tap the in the main toolbar. Your screen will be overlaid with this:



Don't worry about most of this. The only thing you need right now is the button in the upper right that says Create Image Holder. Tap this and a standard iPad image picker will appear. Select an image you would like to mess with. The screen above and the image picker will both go away. Now go to the Controls popup. You will see a new entry in the tray called New Image. Make this the tool shape using the Tray To Tool Shape button . You will see a version of the selected image (how low or high res it is depends on what you have the resolution set to).

As far as IgorAnimation is concerned this object (called an image holder) is just like any other IgorAnimation optical. You provide it with first and second parameter values and it provides you with a color expressed as four numbers. In this case the parameter values are provided by the horizontal and vertical positions and the color is determined by the color of the image at that position. Note that image holders use more memory than any other Igor object since they need the actual image data from their pictures.

There are a lot of ways we can play with this, but we'll start with a somewhat complicated one called a coloring modifier. Find it at Opticals -> Coloring Modifier. Make one of these the tool shape, then open it in the structure.

If you look in the section for the coloring modifier, you'll see an object component labeled Base Optics. This is the optical that is going to be messed with. Assign the image holder from the tray to this component. You'll see that the coloring modifier produces a picture that looks just like the image.

Now let's start altering things. There are red, green, blue, and transparency function object components. These are used to replace the old RGBA values (assuming there are functions in these). But these functions do not use the standard first and second parameters as their parameters. They treat the RGBA values of the color they are modifying as the first through fourth parameters:

Red is treated as Parameter 1.
Green is treated as Parameter 2.
Blue is treated as Parameter 3.
Transparency is treated as Parameter 4.

Now you know why we have so many parameter possibilities in the parameter combiner functions. Let's use one of those now. We're going to change the red value of the optical so that it becomes the average of the red, green, and blue values of the base optics. Assign a parameter average to the red function and turn on the Uses 1, 2, and 3 switches (Uses 1 should already be on). Watch what happens to your image. Now assign a parameter minimum to the green function and turn on its Uses 1,2, and 3. Finally, assign a parameter center distance to the blue function and turn its 1, 2, 3 on. At this point you should have something that is still recognizably your image, but changed considerably.

We're not done with this, however. We just took a batch of colors, put them through modification functions, and produced new colors. We can do this more than once using our new colors to produce newer colors and so on. The first value component in the coloring modifier is called Generations. It's the number of times you feed the colors through the functions. Try raising this by +1 a few times and watch what happens to your image.

If you get a version you find interesting it would be nice to save it, so we're going to go back to the place we made the image holder from in the first place. But first, we need to go high res. Go back to the View controls and slide the res slider all the way to the right. It'll take a bit of time to draw completely. When it does, click again on the camera icon in the main toolbar. When the view pops up tap the Snapshot button. Again, wait a bit. It's making a new image. Type a name for the image in the text box (this will be used as a filename). Then tap the Save button. Your image is now saved in your ipad and can be retrieved in the Apps tab in iTunes when next you synch. You can also e-mail your picture from this window by tapping on the button. It takes some time again, so please wait while your e-mail with attachment appears. Address it where you want and send it. If you choose not to e-mail your picture, tap the Done button. You should probably return the view to low res after all this.

Coloring modifiers are only one form of image manipulation. If you look through the Opticals section of the library you'll find all sorts of ways of creating and modifying opticals. Use the library information to get the details of each optical and play around with them to your heart's content.

If you are primarily looking to use IgorAnimation as an image manipulation tool, it's a good idea to become familiar with the entire Opticals section of the library. There are a large number of objects, each of which can create startling effects. The information button gives an exact description of how to use the optical, but there's really nothing like experimentation to get a sense of what you can really accomplish.

Before we go on to actually building models, there's one minor thing. You may have noticed that we've used shapes and opticals as tool shapes. You can also use functions. What you will see if you assign a function to the tool shape is a graph of the function in the xy plane. This is generally useful, but sometimes, particularly when the function uses two parameters, it's better to nest the function into a coloration (as the red, green, or blue function) or into some other shape (such as a radius of a variable ellipsoid) to really see it in action.

Modeling

IgorAnimation has two spaces in which objects can exist, tool shape space and structure space. The only thing in tool shape space is the current tool shape. Structure space, in contrast, can hold an arbitrary number of objects called characters. Where anything can be a tool shape, only certain kinds of objects can be characters.

We'll build up to what these things are slowly. We'll start with the constituent out of which all characters are made, the body part.

Go to the library and open Parts. The first entry in the new section is called Body Part. Make this the tool shape, then open it in structure.

You'll see a fairly long list of components. We're going to start with two of the object components, shape, and optical. The shape component can have any shape in it; the optical component can have any optical. Assign an ellipsoid to the shape and assign an optical you've made (or just some color) to the optical. You'll see that you have an ellipsoid with that optical wrapped around it.

If you've used a complicated optical you may wonder which points on the shape are colored by which colors in the optical.

The color for a point is the color generated in the optical by the same parameters as those that generated the point in the shape.

This is essentially what a body part is, a single shape with a single optical to color it. To be more precise, it's a shape with an optical oriented in space. Notice that the coloring of the object is not uniform even if the optical is just a single color. To create a 3D appearance the object's color shades toward dark the more the point on the object faces away from the camera.

Before we start moving the object in space, make the radii of the ellipsoid 30, 40, and 50. It will make it easier to see how the body part is turned. For most of what we're about to do, it's a good idea to have the increment at 10.0.

To orient the body part in space we need two pieces of information: where to move it and how to turn it. This information is contained in the body part's compass (two rows down from the optical). Open the compass and you'll see three object components: Position, which can only be a point; Rotation, which can only be a rotation; and COR, which is short for center of rotation, which can only be a point. Open the position and change the X, Y, and Z values as you see fit. As you see, you are repositioning the object in space. Now open the rotation. You'll see the same kinds of angle components that we had with the camera. Here, these angles reorient the body part itself. Change their values and see what happens. Finally, do the same with the COR to see how changing what the object rotates around changes its position and orientation.

There is one more basic component to the body part's appearance: scale. The scale causes the part to grow or shrink by multiplying its size. Try raising and lowering the scale by increments of 0.1 to see what happens.

In practical terms, a body part is a shape with coloring that is positioned, oriented, and scaled in space.

A body part can have other body parts that depend on it, that are essentially parts of parts (Lower arm depends on upper arm, hand depends on lower arm, fingers depend on hand, etc.). In this dependency (called anchoring), each dependent part treats the position and orientation of the part it depends on as if it were the origin. In other words, if you move a part you move every part anchored to it.

To show this, we need to add a new part. There are two ways to do this. The first is to select the new body part row in the body part and assign a body part to it. But there's a shortcut. If you select New Body Part and empty it -- that is, tap the button -- a new body part will be created. The new part will appear under the new body part row.

You can add as many new body parts as you want. New ones appear at the end of the list of previously created body parts. Open this new body part and give it a shape and an optical. Use the compass of the new part to move it around. Notice that it is moving relative to the position and rotation of the part it is anchored to. After a bit of this, go back and move the anchoring body part and watch what happens to the anchored one. Move the anchor and you move the anchored. Now try changing the scales of each. Notice that if you change the scale of the anchoring object it not only changes the scale of the anchored object, but changes the distance between them to maintain the same relative placement.

Store the tool shape in the tray and go back to Library -> Parts. Select Model and make it the tool shape, then open it. There's only one component, labeled Body, and it's a body part. Go back to the tray, select your stored body part, and assign it to the model's Body component. Then select the Tool Shape: Model and store it in the tray.

All a model is in IgorAnimation is a body part and its subparts used as the body of something. Models are mostly a convenience, a way of saying this part is the center of the body, the part from which all other parts extend. If you were making a human as a model, you might make the torso be the body of the model, with the head and limbs being subparts of the torso and so forth.

It's time now to introduce the most basic kind of character in IgorAnimation, the actor. It's also high time to put something in the structure space.

First we need to be looking at that space instead of the tool shape space. Go to the View controls, and tap the word Structure in

. The tool shape will vanish from view. We're now looking at another space entirely with nothing in it.

Select New Character in the structure and assign an actor from Library-> Parts to it. Open the actor and you'll see a bunch of object components, most of which have the word New in front of them, except for the first component, which is labeled Character.

An actor is a group of models arranged around a central model called the character. Any other models in an actor are props or costume pieces. Assign the model you made earlier to the character of this actor. Every actor must have a character, but need not have any costumes or props. You can make any model into an actor's character or a prop or costume piece. To add a model to the props assign it to New Prop; to make it part of the costume assign it to New Costume. Costume pieces and props are treated exactly the same. The two lists are conveniences. If you find it's easier to think of some adjunct models as props and some as costume, use both lists. If you find it easier not to then use one list or the other. Make another model with shape and optical and assign it to New Prop or New Costume to add it to the list.

By default every costume piece and prop is anchored to the body of the character so that they move where the body of the character moves. You can change this by creating anchorings.

Flesh out your character by giving it a few other body parts, then select New Anchoring and empty it using the empty button. This will create an anchoring. Open the anchoring. It has two components: Base Model (which is the model you want to anchor) and Anchor Object, which can be a body part or a model. If you have a prop sword that you wish to anchor to the character's hand, you assign the sword to the Base Model and the hand to the Anchor Object. To do this you have to put both objects in the tray, then assign them to the anchoring from there. If you wish to anchor one piece of costume (like a crown) to another (like a wig), assign the wig model to the Anchor Object and the crown to the Base Model. Notice that the listing of the anchoring says what it is an anchoring of. Note that an object can only be anchored to one other object. The first anchoring in the list of anchorings is the one that will be used.

You might want to rename some of these body parts and models.
Each kind of object from the Parts section of the library can be renamed (Most other objects can't).

Renaming can be done in two ways. For an object in the tray, select the object, type a name in the text box of the controls, then click the name button above it.

You can also rename things in the structure. Select the object, type a new name in the text box in the controls and click the Name button above the Structure table.

Rename things so that you are clear on what you have where.

At this point it might be a good idea to save what we've done. Tap the Save/Load button above the structure and the following will appear:

This panel is used to load or save. Type a name in the text box and tap on the Save button to save. The structure, tool shape, and tray objects are all saved. To load a file, select one from the picker and click on Load.

IgorAnimation save files are XML, lightweight files that are easily transported between systems. Note that if you used image holders then the images they relied on are saved separately with the name IgorSaveImage#. If you are moving these files to another ipad or another computer you will need to move these images as well.

If you left IgorAnimation without saving, your work will still be saved in the LastWork.xml file.

Once you have a file saved, you can also e-mail it using the button. To open a file that has been e-mailed to your iPad, open the e-mail in the mail program, press and hold on the attachment until the popup list of options appears, then click on Open in "IgorIpad".

Once you've saved, modify the actor as you see fit. If you want to look at it from different positions you can move the camera around. Don't use the tool shape camera (since that looks into tool shape space). You should use the camera named Camera (which looks in structure space).

You can have as many actors as you wish, but you might want to have some of them organized together into groups that can move and work together. For this we have the other basic kind of character: the unit. A unit is a group of characters anchored together so they can move and act as one. Put your actor in the tray and from the Parts section in the library add a unit to the list of characters. To review the steps of this:

1. Select New Character in the structure.
2. In the library, click the blue button in the Parts row.
3. In the Parts section, select Unit.
4. Tap the To Structure button at the bottom of the Library popup.

Open the unit in the structure. To add a character to the unit, select New Member and assign an actor or another unit to it. Once added to a unit each of its members become subject to the anchoring of the compass and scale of the unit. Furthermore, since units can be placed inside of units it is possible to create highly complex arrangements and anchorings of characters.

If we had to build every single one of these characters by hand it would be very time-consuming. Fortunately there is a way to copy objects so that new exact duplicates can be made and then modified to vary them. To copy an object (for example, the actor you made before), put it in the tray, select it in the tray, then tap the button. The copy will appear as the new last entry in the tray. If the object you copied was one that can be renamed its new name will be Old Name Copy.

One final thing in this section. You may notice that we don't seem to have any scenery. Actually we do. We just don't distinguish between scenery and living actors. Any object in IgorAnimation changes easily just by changing the components that make it up. In such a situation anything can be alive. By all means make props and scenery, but don't think of them as necessarily static. You can make a building as a single model or as a unit made up of a building and its furniture or make it an actor with the room as character, furniture as props, and wall hangings as costume. Whatever you make you can move and change as you see fit. It's computer animation, so make things animate.

For obvious reasons we have delved into only a few of the objects in the library. There are too many different kinds of shapes, opticals, and functions to cover them all in the manual. Each object has a full description that can be found using the i button in the controls. In building your own models you will find that there are numerous different ways of making things. If you need help or suggestions, use the Forum button in the Help popup. It will call up our online forum in safari. Feel free to ask what you'd like.

Get used to the topics covered here, but don't get bogged down in fine tuning your work. In the next section we'll show you how to make changing the values of components a fair bit easier. We'll then build on that to create carefully controlled puppeted motions which will be usable later on in making movies of your characters in action.

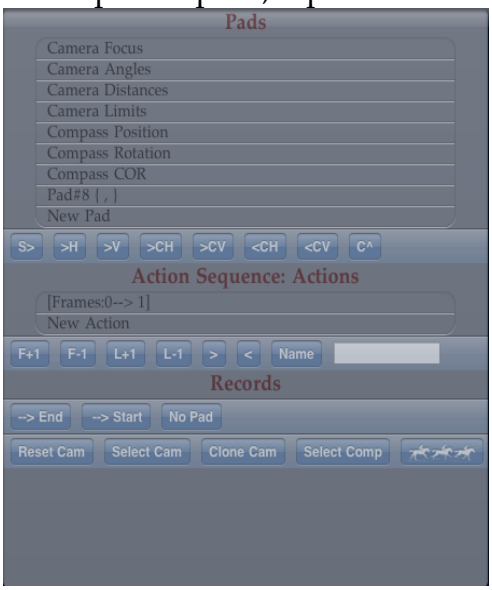
Section 2:
Pads

So far the kinds of things you've been doing haven't been too iPadish. Yes, there's been button tapping and popups, but no grand swooping gestures. Time to remedy that.

The iPad interface is ideal for one of IgorAnimation's interface elements, coincidentally called "pads". Pads are used to set two value components simultaneously by moving one's finger around the view area. One value component is assigned to the horizontal direction, the other to the vertical. In the original desktop version of IgorAnimation pads were squares of screen area that one clicked and dragged in. In the iPad version pads use the entire view. When you use a pad, moving your finger across the view will change those two components depending on where you move. Some pads also have a third component that is affected by a two-finger motion.

We'll start with a very simple example that makes the way pads work easy to see. If you're in the middle of something complex, save your work, then quit IgorIpad and start it up again. Put an ellipsoid in as tool shape and make its radii 30, 40, and 50.

We're going to start by using pads to affect the camera. We'll be affecting the various camera values, starting with the camera's focus. In order to see what effect you're having on the camera, open the tool shape camera, then open its focal point. You don't actually need to do this when using the camera pads; we're just doing so now in order to make clear what changes we're making. To start up with pads, tap the Pads button in the main toolbar. This will pop up:



In the top section (the one labeled Pads), select the row labeled Camera Focus. Notice that the popup button now says Pads : Camera Focus. The pads button will tell you which pad you are working with.

In the camera focus pad, horizontal motion affects the X coordinate (lower at the left edge, higher at the right), and vertical motion affects the Y (lower at the top, higher at the bottom). How much the finger motions change these values depends on the increment. The basic idea is that a range of values is specified with a low value for the horizontal on the left, high on the right, and low vertical on top and high vertical at the bottom.

When you choose a pad, or when you change the increment, the range of values the pad will set is reset to:

Current horizontal value - increment/2 on the left; current horizontal value + increment/2 on the right;

and

current vertical value - increment/2 on the top; current vertical value + increment/2 on the top.

If you set the increment to 100, the range of values you can set for the focal point is

X - 50 to X + 50
Y - 50 to Y + 50

Set the increment to 100. Now drag your finger around the view and see what happens to the values. After a bit, stop dragging. We're now going to fine tune the focal values. We've got focal values on a scale of about 100. Let's do a scale of 10 now. Change the increment to 10.0. Now the pad will drag the values between

X - 5 to X + 5
Y - 5 to Y + 5

The X and Y that form the center of this range are the X and Y values you just set using the larger-scale drag.

You can keep up this process of smaller-scale dragging by simply choosing smaller and smaller increments. In this way, the same big gestures (dragging across the entire screen) can make finer and finer changes. Note: You are not required to keep making the scale smaller, you can also scale up if you want. Or if you wish to keep the scale but recenter the values around the current values, change the increment, then change it back.

You might notice that the focal point has three values and we're only affecting two. To affect the third, do a two-finger horizontal drag; you'll change the Z value. Most pads only affect two values, but camera focus, camera angles, and the compass pads affect a third value with two-finger horizontal drag.

Speaking of camera angles, let's work on those now. Go back to the Pads popup and select Camera Angles from the pads. Reset the increment to 250.00 and start dragging around. Horizontal does first angle, vertical does second angle, and two-finger does spin angle.

You can also use Camera Distances to change the distance and zoom and Camera Limits to affect the horizontal and vertical limits of the camera's view.

If you like the perspective you now have, keep it. Otherwise, tap Reset Cam in the bottom toolbar of the Pads popup to return the camera values to their initial settings.

The camera pads use the tool shape camera when looking at tool space and the current camera when looking at structure space. There will be more about this in the Advanced Topics section.

The next three pads in the list (Compass Position, Compass Rotation, and Compass COR) are used to move objects in space. Select a character or body part in the structure and tap the Select Comp button on the bottom toolbar. This sets these three pads to be the position, rotation, and COR of the object's anchor. These pads have three values just as the camera pads do and use two-finger drag for the third value. If you don't have anything in your structure right now, load a file and test these out.

Pads, as we've seen, can control cameras and object positions. They can also control any other value components you have in your structure. To do that we need to build some changeable pads, as the camera pads cannot be altered.

We already have one changeable pad in the list. It's named Pad#8, which is boring. We'll rename it later. Select Pad#8. The title of the pads button will change again.

Go to the structure. If you have a part whose shape is an ellipsoid we'll use that. If not put an ellipsoid in as tool shape. Select the first radius value component for this ellipsoid. In the Pads popup tap the button labeled >H. Notice that the row for Pad#8 now reads:

Pad#8 {First Radius, }.

This means that the first radius is the value that will be changed by horizontal dragging. Go back to the structure, select second radius, and tap >V in the Pads popup. The row now reads:

Pad#8 {First Radius, Second Radius}.

Our pad is now good to go. We can set the increment and change the radii just as we did the camera values. Let's go with an increment of 50.0. Drag around until you have a shape you like. Then drop the increment to 5.0 to fine tune it. A little redundant reemphasis: because the increment and the starting value determine the range of values the pad can assign, you can get ever finer and finer control without having to use more careful gestures. At an increment of 0.001, the biggest swoop across the screen will still produce only a minuscule change.

By the way, you don't have to have two components in a pad. You can assign only one if you want. Also, which components a pad has can be changed (except for the camera and compass pads). If you like, you can replace the first or second radius in this pad with the third radius. If you need a change of perspective before working on this, select the camera angles pad and move around before going back to Pad#8.

Get used to switching pads. It's often very useful to create a batch of pads, each of which controls two interrelated values. Working on one pad, then switching

to another and back and forth can make it easy to modify multiple values to create a single aesthetic affect.

Meanwhile, that name, Pad#8 is ,as we said, boring. In the Pads popup in the toolbar above the label "Records" is a text box. Type a new name in it and then tap the Name button next to it to change the name of your pad.

In a more complex object (such as an actor) you may have many more things you want to pad around. You can make as many pads as you want. Just select the New Pad entry in the pad list and a new pad will appear (The new pad is always last in the list).

Pads need not have their components come from the same object. You might, for example, want to change a radius in an ellipsoid along with the second angle of a compass. No problem. Pads don't care where their components come from. They just set them.

It can be a bit of a bother to have to keep opening the various parts of the structure that contain value components you might want to place in pads. Fortunately, you can store the value components in the tray. Just select a value component in the structure and tap the same button in the controls that you use to move object components into the tray . The value components appear in the upper section of the tray, the one labeled Binders.

In order to assign a binder to a pad we have to make a slight change in the Pads popup. Notice the button labeled S>. This button means that assignments to the pads are being taken from the structure. Tap this button and it turns to T>, which means assignments are coming from the tray. As long as it says T>, selected tray entries will be used instead of selected structure entries.

One more thing: Your pads will be saved when you save the structure, so that they will all be back when you load a file.

Puppetry

So far what we've done with pads can be useful for refining the shapes and opticals we're making. This is enough if all we're making are still images. In animation, however, pads have a more important use. The changes pads make can be recorded and played back in animations. The pads become like puppet strings. We're going to pull those strings and record the effects, assigning those changes to particular times. We won't be able to play the recordings until later (Section 4, Scripts), but learning to make the recordings belongs in this section.

The object we're going to record the puppetry to is called an action sequence. There's one in the structure already, along with a New Action Sequence entry that can be used to create new action sequences use the button in the bottom toolbar of the structure with New Action Sequence selected to make a new action sequence at the bottom of the list). Action sequences can be named, which is a

good idea if you have more than one of them. The Pads popup is set up at the beginning with the initial action sequence in it waiting to be puppeted into. To change which action sequence you are recording to, select an action sequence in the structure and tap the in the bottom toolbar of the Pads popup.

An action sequence consists of a sequence of actions (no surprise there). Each action consists of a pair of recorded values for value components: the values are the initial and final values of those components. In playing an action the value components change from their initial values to the final values. The actions of the selected action sequence are found in the second section of the Pads popup, the section labeled Action Sequence Name : Actions. There is a single action in the sequence at the moment. It shows up in the list with the name [Frames :0-->1]. We'll come back to the matter of frames later. You can name the action if you want (it's not really necessary but it can help in a complex sequence). Select the action, type a name in the text box and tap Name. Notice that not only does the name change in the section but the section below is now titled Name : Records.

To record values, select a pad and use it to set up the initial way you want its components to be (for example, you might pick Camera Angles and set up the beginning angle values). Then tap the -->End button in the toolbar below records. What you will see in the records is each of the components of the pad listed with followed by (current value, current value). Now, use the pad to change those values, then tap -->End again. Notice that the second values in the parentheses have changed, but the first values are the same. You have just recorded a puppeted change, with the initial value you set from the first tap to the final value you set from the second.

However, neither of these values is written in stone. You can reset the initial value from the current pad values by tapping --> Start, and reset the final values by tapping --> End.

You can change pads and record the values of that pad in the same action. Indeed, a single action can hold as many value different components as you want. But each component has only a single initial value and final value in that action. Notice that because values are only recorded when you tap the -->Start or -->End buttons, you can do as much fine tuning as you wish, as much careful arrangement of beginning and end conditions as you like, before recording them to the action. The changes you make as you move your fingers are not recorded, only the beginning and end. You can take as much time as you need to refine your initial and final values.

Once the first action has the changes you want to record, you can make a new action and record into that, with new beginning and end values for any components, including any of the ones in the last action if you want to.

Standard practice is to start one action where you left the last one off (the ending values from the last action are used as the beginning values for the next). This creates the appearance of seamless motion.

The only thing that has not yet been set by what you have done is when in the playing of this action sequence these actions occur. That's where the frames come in. Actions occur at specified times with specified durations. At the beginning time the components in the action are set to their beginning values; at the end they are at their end values. In between they take values in between proportional to the time between beginning and end. The beginning and end times for each action are specified by a first and last frame number. How long a frame is in actual time depends on your clock (this will be discussed in section 4). Default in IgorAnimation is 16 frames per second. You set the frame numbers for the selected action using these buttons:

The first and last frames for an action are the numbers after the action name:

Name [First Frame --> Last Frame].

F+1 moves the first frame up 1, F-1 moves the first frame down 1. The actions in the sequence are sorted by first frame, so as you move the first frame up and down you may find the action swapping places in the sequence. Note that raising the first frame raises the last frame by the same amount. L+1 raises the last frame number, L-1 lowers the last frame number (the last frame cannot be less than first frame + 1). The last two buttons, > and < , raise and lower the increments on the previous buttons by a factor of 10.

Using the frame values you can control exactly when and for how long each of your puppeted actions take. You can also coordinate the actions down to the single frame.

That's all for the uses of pads until we cover playing back the actions.

Optical Zooming

There's one other application of dragging across the screen that doesn't involve pads: the ability to zoom in on regions of a drawn optical.

Zooming to look at shapes and characters is done with the cameras. But opticals don't respond to the camera. They just get drawn, filling the entire view. However, there is a way to zoom in on a particular part of the optical.

There are two requirements to use this:

1. There must be no selected pad. If you have a selected pad, tap the No Pad button in the toolbar below the Records section in the pad view.

2. The Zoom switch in the View controls must be on.

If the Zoom switch is on and there is a selected pad, dragging across the screen still activates the pad.

The easiest way to see how the optical zoom works is to create an image holder and make it the tool shape. Make sure tool space is selected for viewing. Low res would be a good idea.

The first part of optical zooming is to pick a center for the region you want to look at. To do that, double tap on a part of the image. The image will shift around to be centered there.

To zoom in or out tap, hold, drag and release. Dragging upward zooms in vertically; downward zooms out vertically; dragging left zooms in horizontally; right zooms out horizontally.

Zooming in and out are done proportionally to the current zoomed area.

To restore everything to normal, triple tap anywhere.

Zooming in has three major uses:

1. Examining your creations.
2. Watching the effects of changes (from setting values or pad use) in a small area.
3. Chopping opticals up.

This last involves the use of a new kind of object: the range. Ranges can be found in the library in Shapes -> Points, Rotations, and Ranges, but they can also be created by the zooming we've been doing. Zoom in on an area of your optical. Then go to the View controls and tap the + button next to the Zoom switch. If you check the tray you'll see a new object labeled Range. Zoom out to the full image, and turn zooming off.

Now go to the library and open Opticals -> Combined & Modified Optical. About halfway down the list is Range Limited Optical. Make this the tool shape then open it in the structure. There are two components: Base Object and Range. Assign the optical you were zooming in on before as the base object and the range you just created as the range. You will see an optical that is transparent outside of the range you zoomed in on and is the cut off area of the range inside it.

You can get more or less the opposite effect (fitting an entire optical into a smaller area), by using Range Filling Optical. It has the same component names, make one of these the tool shape and again assign the image holder to the base object and the range to the range.

Details on ranges can be found by using the i button as usual. Ranges can be quite complex, and can produce far more complex slicing effects then the simple ones we just created. To examine a range directly in order to see what area it slices out, make it the tool shape. You will see a dynamic optics cut down to the area the range makes visible.

There are also range limited and filling shapes and functions, which carve pieces out of shapes and functions.

Section 3:
Snapshots and Images

It's time to start putting the pieces together to make images from these objects.

The full viewable structure consists of more than just structure space. There are really three layers:

1. Back optics, which can be any optical
2. Structure space
3. Front optics, which can be any optical.

Structure space is drawn in front of the back optics, and the front optics is drawn in front of structure space. If the front optics is fully opaque you won't see anything of structure space or the back optics.

Back and front optics are not affected by the camera, but they are affected by optical zoom if it is turned on.

What you see of structure space is not affected by optical zoom but is determined by the camera.

For back optics to be visible, there must be an optical assigned to back optics and the Back Optics switch in the View controls must be on. Similarly for front optics and the Front Optics switch.

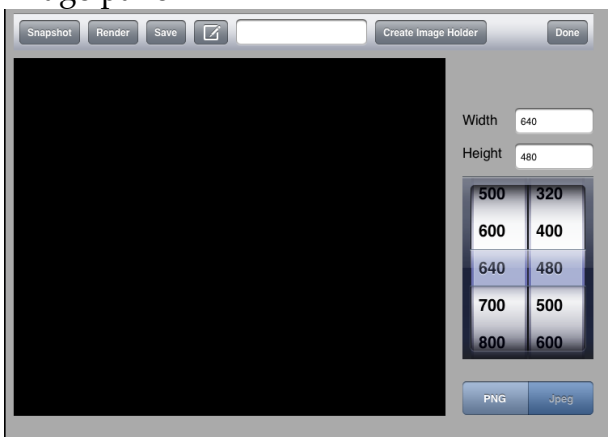
Note: You can have a front optics and a back optics with nothing in the structure space, in which case the front optics is simply drawn on top of the back optics.

Having both front and back optics on will slow things down a fair amount. It's a good idea to have them both on only when you need them both on.

Because of this arrangement you can use IgorIpad to combine 3D and 2D or just work with one or the other. Decide on something you'd like to make a picture of and build it or load it.

You can make an image from either the tool or the structure, and you can make an image at any resolution. Good images should be done at high res, so set things up at low and medium res, then switch to high res. When the screen has

finished redrawing, tap the camera icon in the main toolbar to bring up the Image panel:



Before you actually create the image you must specify the width and height of the image you will draw. You can type in the Width and Height boxes, or if you are using standard values, use the wheel picker (the first wheel sets width, the second sets height). Note: the image you make will not automatically have the same aspect ratio as the screen.

The next decision is the type of image. PNG and JPeg are your choices. Use the segmented control beneath the wheels to choose file type.

To create the image you have two choices, snapshot and render. If you are only drawing opticals these two work exactly the same way. If, however, you are drawing IgorAnimation objects (characters in structure space) they do not work the same. The snapshot uses the same shape approximations that drawing on the screen uses. Render uses a slower but more accurate method of drawing. It's a good idea to use snapshots for testing and renders for final images. Tap Snapshot or Render to start the image making. When the drawing is done it will appear in the image view area below the toolbar. Once the image is created you can save it if you like by typing a filename in the text box, then tapping the Save button.

Once the file is saved you can, if you wish, e-mail it by using the standard email button next to the Save button. Warning: High resolution pictures can take a while to attach to e-mails. Be patient. E-mailing will close the Image panel.

If you do not e-mail you can close the Image panel at any time by tapping the Done button.

Once an image is saved it also becomes available to be used in image holders. Once you've made your snapshot or render, you can tap the Create Image Holder to select the image you just made and start messing with it inside IgorAnimation.

Section 4
Scripts and Animation

To create animations IgorAnimation uses IgorAnimation scripts. Just as IgorAnimation objects are essentially instructions on how to draw themselves, scripts are instructions on how to change objects over time before they draw themselves in each frame.

A script consists of one or more script lines, each of which is a single instruction for changing a single aspect of something. Each script line can have child lines which contain their own instructions and possibly their own child lines. Each script has a top line which is not the child of any script line.

What a script line does depends on what kind of script line it is. A move line will move an object in space; a binder line will change the value of a single value component. Information about each line type is available in the Script popup (see below).

Before we get into script building we have to discuss the subject of time. Time in IgorAnimation is set by the clock. You'll find the clock in the structure. Yes, that New Clock line means that you can have more than one clock. That's an advanced topic we'll cover later in section 5.

Open the clock and take a look at the value components:

Time represents the time the clock is currently set at. When animation is being done, the time is incremented each time a frame of animation is created. Time is measured in seconds.

Tick represents the amount of time the time is incremented by when the animation is being played. When movies are made from an animation, the length of time each frame lasts is equal to the tick. Tick is measured in seconds.

Begin is the time the clock starts at.

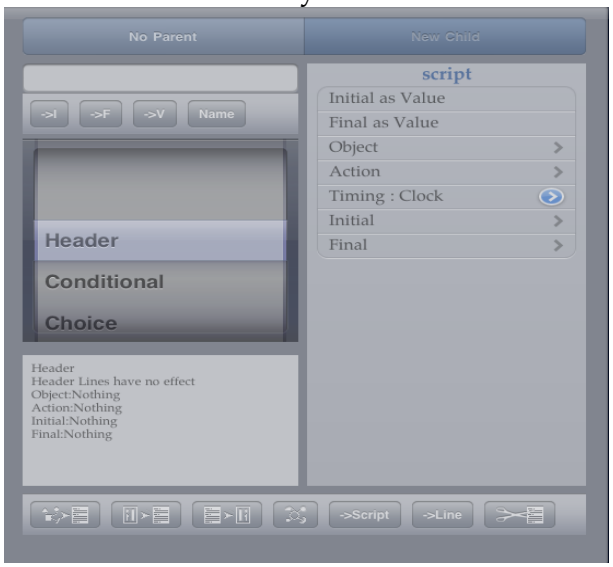
End is the time the clock ends at.

You can change any of these as you want. There is not much point in changing Begin. Change End if you want the clock to run longer, to make longer animations.

Frame Rate is the number of frames per second for the clock. This is used in two ways. First, it connects time in frames, which action sequences use, with time in seconds, which scripts use. Second, when you change the frame rate you change the tick to 1.0/frame rate. But the opposite does not happen. Tick can be set without resetting the frame rate. You can make the clock skip over as many frames as you want when actually doing animation.

Along with their timekeeping, clocks can also be used as functions. The value they produce is called their normalized time, which is (Time - Begin) / (End - Begin). This is usually a number between 0.0 and 1.0 (0.0 at Begin and 1.0 at End). The normalized time of the clock is used in a variety of ways by scripts.

To start making scripts we need to open the Script popup by tapping the Script button. Here's what you should see:



The right-hand side of this popup has a table that acts like a section of the structure showing a single object (a script line) with its components. Its object components can be opened and modified just as components in the structure can. In particular, what it is showing now is the object listed as "script" in the full structure.

There are five components in a script line, but they masquerade as seven. Most script lines use only two or three of them:

Object: The object being affected by the script. Depending on the line type, all sorts of things can happen to the object.

Action: The object that affects the object. Depending on the line type, this might be a shape, a function, an action sequence, etc.

Timing: The clock that determines when things are happening. Every script line has a timing. By default it's the standard clock in the structure (called the base clock).

Initial: Some line types require or allow an initial value or initial object. The Initial as Value line shows it if it's a value; the Initial line shows it if it's an object.

Final: Some line types require or allow an final value or final object. The Final as Value line shows it if it's a value; the Final line shows it if it's an object.

The line type for the script line is set using the spinner to the left. When you select a line type the explanation of what the type is and how it works appears in the text area below the spinner.

We'll start by doing something really dull. We're going to make a one-line script that will cause an ellipsoid-shaped actor to move in a circle. We need an actor with a character which has an ellipsoid for a shape and a color for an optical. Please make it; we'll wait.

Once it's made, select it in the structure We also need an ellipse for it to move around. Put an ellipse in the tray and select it in the tray.

The line we're going to make is called a move line, so select Move in the spinner in the Script popup. An explanation of the move line will appear below the spinner.

A move line needs an object that is the thing to be moved. Select Object in the script table, then tap the button (which assigns from the selected structure entry). The actor you selected has just become the object of this move.

A move line needs the curve to move along as its action. So select Action in the script table and tap the button (which assigns from the selected tray entry). The ellipse you created is now the action of this script.

That's all you need to make a workable script. To play it, we need to go to a different view. Dismiss this popup and tap the button in the main toolbar. This Play Script panel will appear:

Some of this is similar to the Image panel. In particular, one can set the resolution of the movies one is making, save them and e-mail them. There will be more about this later.

Let's make a movie. Press the Start button in the toolbar. The progress bar right after the Start button in the toolbar allows you to watch the movie being made.

The script will have as many frames as there are ticks between the Start value and the Finish value (more about this later). Once it's done, your movie will appear in the center area with standard movie controls.

Should the program quit during this process the already-made section of the movie will be saved in a file named recover.qt. Hopefully, your full movie was made. Play it as long as you find it interesting. The + and - buttons on the top bar are used to move the movie one tick forward or back. When you've had enough watching, tap Done.

So what just happened?

The move line takes the curve assigned to the action and uses it to set the position of the object. As the clock ticks from start to finish, the script line calculates the point on the curve determined by setting the first parameter value equal to the normalized time on the clock. The structure is then redrawn and used as the frame corresponding to that time. The process of carrying out the instructions of the line is called enacting the line. Each line in a script is enacted once each time the clock is ticked.

The move line type is one of the three most common in use. The other two are binder lines and function binder lines. Both of these use a binder (value component) as the object and are used to set the value of that binder as the script is played.

A binder line simply sets an initial and final value for that binder. Enacting the binder line sets the value of the binder to Initial * (1.0 - normalized time) + Final * normalized time. At the begin time the binder will be set to the initial value and at the end it will be set to the final value. To set an initial value type a number in the text box and use the >I button. To set a final value type a number and tap the >F button.

Function binder lines use a binder as object and a function as action. When this kind of line is enacted the function is evaluated with the normalized time as first parameter and tick as second parameter. The binder is set to the resulting value.

Let's go back to the Script popup. So far we've only made a one-line script. It's pretty easy to make more lines in a script. Each script line has at most one parent line and as many child lines as one might like.

Before we make more lines, it's a good idea to name this one. Type a name in the text box and tap the Name button. Now notice the bar at the top of the popup. It has two sections right now labeled No Parent (because this line has no parent) and New Child. Tap New Child.

A new section has been added in the middle of the bar labeled 1:script. Tap this and you'll see a fresh new line appear with the line type Header and empty components waiting to be set. Also notice that the bar above now has Parent: (whatever you named this). Using this bar it is possible to create new lines (with

New Child) and navigate up and down the generations of the script by selecting parent or child entries. You can make a script as complex as you want, nesting script lines as children in other script lines.

One important thing. When script lines are enacted they are enacted in a specific order. A parent is enacted, then each of its children in the order given in the bar above. So if you have

Parent:
    Child 1:
        Grandchild 1:
        Grandchild 2:
    Child 2:
        Grandchild 3:
    Child 3

The order given above is the order in which the lines are enacted:
Parent, Child 1, Grandchild 1, Grandchild 2, Child 2, Grandchild 3, Child 3.

This can matter a great deal if multiple lines are affecting the same value. For example, if one line moves an object somewhere and another puppets the X coordinate from the object's compass, then the final position of this object is determined by whichever of these lines was last in the ordering.

Also note that certain kinds of lines (particularly the Function Bind+, Function Bind*, and Function Bind Iterate lines) require that a value have been set by something else before they modify it. These should always follow such a line in the enactment order.

One line type in particular is of special import: the puppetry line. A puppetry line needs an action sequence as its action and no other component (apart from timing). The puppetry line plays out the recorded changes of the action sequence, matching frame for frame against the frame rate of the clock. Frame 0 corresponds to time 0; frame 1 to time 1/frame rate; frame 2 to time 2 / frame rate and so on.

You can dispose of a line you don't like by making it the active script line and tapping on the trash can. The line will go into the tray, in case you didn't mean it.

There are also two ways to pull scripts out of the structure. If you select a script and tap ->Script, that script will become the top script in the script view (the script without parents). If you click on ->Line you will make the selected script a child line of the current line. In this way you can assemble complex scripts out of simple ones.

Play around with the script lines. Read the descriptions for each line type by selecting the line type in the spinner. Make puppetry lines with action sequences you've made as the actions. Make binder and function binder lines for various

value components you want to change. Be aware that the moveturn line is a very complex way of reorienting an object in space. Unless you need to do things that complicated, we recommend using binder and function binder lines to set the angles or action sequences to orient the objects.

When you have a script you'd like to see in action, we can go back to the script player.

Are we back? Good. A few things about the script player. Whether the script draws normally or renders is determined by the Standard/Render control. The width and height of the movie you are creating is set by the width and height boxes, or you can use the spinner for standard values. You can change the tick and frame rate (using the Tick and FPS boxes) as well as the begin and end values here. The start and finish values you can set in this panel are when to start and when to end playing. You may have made a script that would generate four hours of animation, but that doesn't mean you should do it all at once. It's better to do animation in short bursts of time and assemble full movies later with tools like iMovie. Use start and finish to set when within the bounds of the clock you wish to animate. If you want to animate from second 5 to second 8.2, set start and finish accordingly. Note: finish cannot be higher than the end time for the clock, nor can start be lower than begin.

When an animation has been made, you can save it by typing a filename in the box and tapping Save. You can then e-mail it if you want using the e-mail button. Movies created are in QuickTime format.

One more thing. The scripts listing in the structure is not the only place you can store scripts. You can have scripts as components of body parts and characters, making them motions of that body part or character. Each of these kinds of objects has a new motion component in their listings. Assigning a script to one of these will make it a motion of that object. Emptying the new motion will create a new script as part of that object. These scripts are usable just as any other script, but they can also be accessed using the named motion line type (see the line type descriptions in the Script popup) and named motion attributes (see the advanced section below)

Section 5:
Advanced Topics

Styles

The appearance of every IgorAnimation character is determined by the shape, optical, substance and orientation of the objects. But suppose you want to change how something looks without changing what it is. There are two ways of doing this in IgorAnimation: styles and lighting (which will be discussed below).

Styles are found in the Styles & Lighting section of the library.

There are three kinds of styles: styles, which are the basic style object;

multistyles, which allow you to choose between a number of different styles; and morphstyles, which morph between the effects of two different styles. To create a new style assign a style from the library to the New Style component in the structure.

Styles are special objects that contain several different components that modify the appearance of an object on the screen. Whether these components are employed depends on whether or not various switch components of the style are on or off.

1. Position Modifier. This is a shape modifier that changes the positions of the points of the object being styled. It is generally used to distort the appearance of things. Position modifier is active if the Modify Position? switch is on.

2. X Function and Y Function. These modify the projected points -- that is, the points on the screen that the points of the styled object are projected to. X Function and Y Function are functions that use horizontal and vertical position as X and Y parameters. The positions are relative to the center of the screen, with the center being (0.0, 0.0). The numbers these functions operate on are measured in projected distances not scaled to the parameter square. Note that moving the camera can radically alter the effects of this style option, since these functions care only about where a point seems to be. These functions are only active the Modify Projection? switch is on.

3. Color Modifier. This is a coloring modifier used to change the colors of each point on the object. Note that the color this modifies is the final color of the object (including substance and shading). This is used if the Modify Color? switch is on. The base optics of the coloring modifier is a placeholder color used in modifying other colors. Don't change the base optics. If you substitute some other coloring modifier as the color modifier, place a color in its base optics. Because lighting also modifies the color of the object, there is a question of whether the color modified by this component is the prelighted or post-lighted color. If the Lighting Before Style? switch is on, lighting is done first. Otherwise it's done after the style is applied.

4. Lighting Modifier. This is a coloring modifier used to change the lighting color that falls on the object. This is used if the Modify Lighting? switch is on. The base optics of the coloring modifier is a placeholder color used in modifying other colors. Don't change the base optics. If you substitute some other coloring modifier as the color modifier, place a color in its base optics.

Styles also have a substyle component which can be any other style. Substyle effects are applied before the effects of this style. Note that while this style might not modify some aspect of the object's appearance, the substyle might.

Styles can be assigned to two different kinds of objects: body parts and units (each of these have style components).

The style affects the object and every object anchored to it. If you assign a style to:

1. A body part, it affects the child parts of that body part and their children and so on. Furthermore, if the body part is in an actor the style affects every object anchored to that part. Note that since by default every prop and costume piece is anchored to the top part of the actor's character, every prop and costume piece not specifically anchored elsewhere will be affected by that style.

2. A unit, it affects all the members of that unit and all their parts as well.

However, if you assign a style to an object that is being styled by some other object higher up in the anchoring chain, the new style pre-empts the old one. For example, suppose you have a unit representing an army and you assign a style to it that makes the members of the army smaller and greyer. But one member of the army is an actor meant to be some person in the blatantly heroic line. You could assign to the body of the character of that actor a style that makes him and all his props and costume bigger and brighter. The unit style would not affect that actor. If you further decided to make one of the actor's props (like a sword) dark and grim, you could assign a style to that that preempts both of these previous styles.

Using styles in this way it is also possible to use multiple copies of the same model and have them appear wildly different in actual viewing.

Note that styles only do their modifications if the Styles switch is on in the View controls. Using styles slows down drawing a bit, so do as much as you can without them before turning them on.

Lighting

We frankly recommend styles as the means to modify appearance of objects, but lighting is available for those classically minded. Lighting objects in IgorAnimation are found in the Styles & Lighting part section of the library. Lighting is a single light source. Multilighting is a light source that can take multiple forms. MorphLighting is a light source that morphs between two different lighting aspects. To create a new lighting object, assign a lighting from the library to New Lighting in the structure.

Each lighting has a switch component called On? If On? is on the light is on. Otherwise it isn't.

Each lighting has two object components:

Source, which can be a shape or a point

Illumination, which can be any optical.

Source and illumination are somewhat complex objects. They are respectively used to produce the direction of lighting and the color the lighting sheds for each point in space. They use position in space as the parameters. X is first parameter, Y is second, Z is third.

The light cast on a point of an object by the lighting is determined by drawing a line from the source for that point and assuming that light the color of the illumination is shining on it. Opaque objects in the way will block the illumination. Translucent objects will change the illuminating color the way translucent objects change illuminating colors.

Note please that the directions of lighting need not make sense in real-world physics, particularly if you are using a weird enough shape for source.

The color of the point on the object is determined by additively mixing all the lighting colors on that point (it is solid black if no light falls on it), then subtractively mixing the resulting color with the actual color of the object at that point. A nonilluminated point will therefore be black.

Lighting is only used if the Lighting switch in the View controls is on. And an individual lighting will only contribute to the actual lighting if its On? is on.

Attributes

So far the various value components in objects have been set one at a time (or two at a time in a pad). Sometimes, however, you want values, shapes, and opticals to have some basic underlying coordination, an invisible relationship that allows them to change in harmony. For this IgorAnimation has attributes. Attributes can be very complex to use and they are not necessary for standard usage of IgorAnimation. They exist for large, complex interacting structures.

Attributes can be found in the Attributes section of the library. Each attribute is assigned to a body part or character using the New Attribute component of the body part or character. When you make such an assignment, that attribute becomes an attribute of the object. Each attribute has a name and can be renamed using the standard rename methods. It's a good idea to give each attribute a name related to its purpose.

Each attribute can be used in one of three roles:

1. As a function. As a function the attribute has a (usually constant) value which it evaluates to.

2. Set value. Each attribute can take the place of a binder in a value-setting situation (for example, any attribute can be made the >V or >H component of a pad or the object in a bind or function bind script line). This allows the value of the attribute to be set by the value-setting situation (for example, moving around the view when the pad is active).

3. As rules to be applied. See the discussion of rules later).

The simplest attribute is called Attribute. It's essentially just a named value. To illustrate its use, get one from the library and make it the first radius function of a variable ellipse. Then put the attribute in as the >H of a pad and change it's value by dragging. As you see, the attribute's value is changing and the ellipse is growing and shrinking accordingly.

One of the two most useful attributes is called a calculated attribute. This is an attribute with a base attribute (which can either be an attribute or a binder) and a calculator function. When the value of the attribute is set, the calculator function is evaluated with the value as first parameter. The resulting value is then used to set the base attribute. To set this up, make an actor and put it into the structure as a character. Make the shape of its top body part an ellipse, and use a color for its optical. Put the first radius of the ellipse into the tray as a binder. Now select calculated attribute from the library and assign to the New Attribute row of the actor. Open the calculated attribute and assign the first radius from the tray to the base attribute of the calculated attribute. Put various functions in as calculators and see what happens as you set the value of the calculated attribute.

The other most useful attribute is called a list attribute. A list attribute has a list of attributes which can be added to by assigning an attribute to the New Entry component. When the value of a list attribute is set, the values of each of its entries are set to the same value.

If you make a list attribute whose entries are calculated attributes, you can set multiple value components each according to their own functions with a single value. This allows for the setting of multiple values in harmony. You could, for example, change the shape and color of an object simultaneously with a single value setting (using a binder script line or a pad, for example).

One way to think of list attributes with calculated attribute entries is as very sophisticated puppet strings.

Bear in mind that a list attribute can be the base attribute of a calculated attribute, allowing for nesting of puppet strings in other puppet strings. Just using these two attribute types you can make phenomenally sophisticated changes with a single value setting.

Another useful attribute for setting is the bounded attribute. This has a base attribute as well, but it has lower and upper bounds on values. If you try to set its value to below the low value it sets the low value instead; similarly for setting a value above the high value. Using a bounded attribute you can make sure you never set the base attribute's value too large or too small. This can be very useful in, for example, crafting the motions of joints in animal limbs.

Attributes as rules: Attributes can also be set to do specific things when explicitly applied. In this case the attribute acts as a list of instructions,

modifying one or more value components. Attributes set like this are called rules. Each attribute in the library has an As Rule entry in its library entry.

There are three ways to apply an attribute as a rule:

One is by making the attribute the action in an apply rule script line (in which case the rule is applied every time.

The second is by making the attribute the setup rule in an action sequence. The setup rule is applied when the action sequence is started.

The third is by the use of the Apply button in the controls. If an attribute is selected in the tray, this button will apply it. One good use of this is to create a List Attribute, Rule Sets Vary that will set a batch of values back to starting conditions, so that if you need to reset things in the midst of experimentation you can.

Named attributes and attachment to objects: There are three very special attributes, called named attributes. A named attribute has a base object which is usually a body part or a character. The named attribute has a part name which is simply a name you type in.

A named attribute is treated as if it is the attribute of that name belonging to the base object. For example, if the base object is a unit with an attribute named "Morale" and the part name is "Morale", then the value of the named attribute is the value of the unit's "Morale". Setting the named attribute's value sets the attribute of that name in the base object.

A named body part attribute uses its name to act like the part of the base object that has the part name. In this case, body part has a more general meaning. The props and costume and character are body parts of an actor and the members are body parts of a unit. You can, for example, use a named body part attribute as the object in a move script line and you will move the part that is named.

A named motion attribute acts like the motion (that is, script) of the base object that has the same name as the part name. You can use a named motion attribute as the action in a NamedMotion line.

If you are going to use named attributes, make sure you give explicit and distinct names to each attribute, part, and motion in a given object.

Attributes can be attached to objects, which essentially means that the attribute will be affecting that object. In practical terms this means that named attributes will use that object as their base object and other attributes will attach attributes contained within them (like the entries in a list attribute) to that object.

There is an Attach button in the top toolbar of the controls. This attaches the attribute selected in the tray to the object selected in the structure.

There are also two buttons called MApply and MApplyR. MApply is short for MultiApply. When you multiapply an attribute to an object, the attribute attaches to each body part of the object, then the attribute is applied as a rule. MApplyR does a recursive multiapply. Not only does it apply to the parts of the object, but to their parts as well and so on down.

Note that there are attributes to do attachments and multiapply as well.

AI block attribute: The AI block attribute is the most complex attribute in IgorAnimation. It is used to create a semblance of intelligence, or at least reaction in a character.

This attribute is complex in application and values.

Here is the application process:

The AI block attribute has a list of bounded attributes called its ranges. When the block is applied as a rule, it first tests each of its ranges. If the base attributes of the range are within its bounds (value greater than low and less than high), then the block passes its range test. The number of ranges that must pass the test for the block to pass is equal to the block's threshold value component. If the threshold is 0 or less, then all the ranges must pass the test.

The block has a list of attributes called its rules. If the block passes the range test then its rules are applied in the order they appear in the rules list.

The block has a list of AI block attributes called its blocks. If the block passes the range test then its blocks are applied in the order they appear in the blocks list.

Essentially, this means that the block tests for various conditions defined using the base attributes of the ranges. If enough of those conditions are met, the block's rules are applied and then its blocks are themselves tested and applied. This allows for very sophisticated testing for which rules to apply and which to ignore.

If the Set By Fit? switch is on, then assigning a value to the AI block attribute will assign that same value to its rules and blocks if its ranges pass the test.

The value the block evaluates to when used as a function depends on the value accumulation type, which can add up all the values of all the rules within all the blocks that meet their tests, or average them or take maximum or minimum. In effect one can get an intelligent value response as well as set values intelligently.

AI block attributes are only necessary in very sophisticated uses of IgorAnimation, but in those uses they are extremely useful.

Pads

Pads normally calculate their values using the range specified by the increment. However, you can also give a pad two functions (CV and CH) used to calculate the values for the V and H Components. These functions can work on up to 4 parameters:

Parameter 1 is the horizontal position you dragged to.
Parameter 2 is the vertical position you dragged to.
Parameter 3 is the current H value.
Parameter 4 is the current V value.

To set CH use >CH, from either the structure or the tray (depending on whether the S> or T> button is being used). Use >CV to set CV. To extract these and put them in the tray use <CH and <CV.

Setting the calculator functions does not automatically make them the means of determining the values for that pad. Tap the C^ button to use these alternate calculators. Notice that a ^ appears at the end of the pad name in the popup button. Tap the button again to go back to the normal increment-based calculators.

Note: The alternate calculators are not affected by changing the increment.

Cameras

It is possible to have more than one camera looking at structure space. There are two ways to create a new camera:

1. If you empty the New Camera entry in the structure you will make a new camera that is looking straight down from a height of 100.0.

2. If you tap the Clone Cam button in the Pad popup, you will make a copy of the current camera with the same camera values (focus, angles, etc.) that the current camera has. You can move a camera around and make multiple clones as you find useful vantage points on your structure.

To choose a new camera as the actual camera being looked through, select the camera in the structure and tap Select Cam in the Pad popup. The camera you chose will become the new camera.

Note: Cameras can be renamed. If you use multiple cameras it's a good idea to name them.

When you load from a saved file, the first camera in the camera list will be the active camera.

Clocks

It is possible to have more than clock. To create a new clock, empty the New Clock component in the structure. The original clock is called the Base Clock, although it's name in the structure is Clock, and it is used to determine the beginning and end times of script play, as well as how many frames a played script will have. But other clocks can be used as the timing in script lines and can be used to produce more complex effects with scripts.

When a script plays, each frame it goes through is a single tick of each clock used as timing. That means that a clock with a different tick and/or different begin and end values will produce different normalized time values and therefore will produce different effects with each script line since they almost all depend on normalized time.

A script line does not do anything if the time is before its timing's beginning or after its ending. This means that a script can contain lines that only enact during part of the entire script run.

A clock can be set to cycle by setting its Cycles? switch to on. A clock that cycles starts over at the beginning when it reaches the end. If you have a cycling clock in a script it will run over and over its actions again and again because its normalized time will cycle from 0.0 to 1.0. Each time the clock cycles back to its beginning and goes through to its end. This allows for repetition of action during a script. But the repetition need not be exactly the same. For example, if a value important to a script line is being set in another line that does not cycle, you can create cycling with variations. Do not set the base clock to cycle as this will cause problems in script play.

A clock can be made a much more complicated object by the use of a rhythm. The rhythm of a clock can be any function. The function is used to determine how much the click ticks each time a script line enacts. Don't put a rhythm in the base clock.

In a rhythmed clock the tick value is determined by the rhythm function using the following parameters:

First Parameter: Time
Second Parameter: Tick
Third Parameter: Begin
Fourth Parameter: End.

If the Rhythm in Frames? switch is on, then the rhythm function calculates how many frames to tick by rather than how many seconds.

Note that rhythm values can be negative so a clock can move backwards or jump around.

Website, Other Versions, and Further Help

The official IgorAnimation website is: http://www.igoranimation.com/
The site has, among other things, forums for IgorAnimation discussion. Users are urged to contribute their ideas and effects. We also welcome suggestions for changes to the app. The Forums can also be reached by tapping the Forum button in the IgorIpad Help popup.

IgorAnimation has existed in earlier forms on the Mac. At present a cross-platform version is under development which will use the same file types as the iPad version, allowing for full cross-compatibility.