The Basics of Animation

Robert Bridson

September 6, 2011

1 Perception

Essentially all animation is a series of still images, called *frames*, displayed rapidly in sequence. If shown fast enough, the human visual system may interpret this as smooth motion.

The psychological research into how exactly this works is ongoing: the keyword to look up is *apparent motion* or the *phi phenomenon*. It has been subdivided into special cases such as *beta* (where showing a still object at rest in different positions in sequence, when done not too fast and not too slow, leads to the illusion the object is moving) or *phi* (generally involving a faster rate of change where motion is perceived without any sense of the shape or form of what is doing the moving). In fact, motion can sometimes be perceived in a single still image (*illusory motion*), probably correlated with the motion of the viewer's eyes themselves. You may well have also encountered the situation where after staring out of the window of a moving train for a long time, the scenery appears to move in the opposite direction once the train comes to a stop.

Research into human vision continues to unravel how we perceive the world, with many experiments involving optical illusions that tease apart what's going on in our vision system. It is clear that what we consciously think we are perceiving often is largely interpolated by the brain (or earlier structures in the optic nerve) from much sparser data. This has obvious evolutionary advantages: being able to "see" a single predator moving behind dense bush, instead of a multitude of disconnected snippets moving between the leaves, is one example where apparent motion has a huge benefit.

At any rate, it is enough to know for the purposes of animation that we can exploit this evolved capability to fake continuous motion with a rapid sequence of still images. The next question, of course, is what those still images should be. Perhaps one day increased knowledge of human vision will lead to a definitive answer, how to optimize those images to produce

the desired effect in the viewer. Right now, we settle for simply mimicking a real motion picture camera, as we know it works pretty well: taking a sequence of images and then playing them back gives a very good impression of the continuous motion that actually took place.

1.1 Persistence of Vision and Motion Blur

Regrettably, there is a continuing confusion about apparent motion and an effect called *persistence of vision*. Persistence of vision refers to the fact that when a light receptor in our retina is exposed to light, it continues to send signals to the optic nerve for a while even after the light has stopped. At the most basic level, this is a function of the chemistry of our eyes: there are light-sensitive pigments such as *rhodopsin* and various *photopsins* in our photoreceptor cells which change configuration upon exposure to light, triggering neural signals. It takes a while for enzymes to "bleach" these chemicals back to their normal state, when neural signals cease.

One of the consequences of persistence of vision is that very rapid pulses of light cannot be distinguished from continuous light by our eyes. For example, many fluorescent lights actually flicker at 120Hz, but generally appear to be emitting constant light.[1]

Traditional film projectors also exploit persistence of vision. When moving from one frame to the next, the light is temporarily blocked by a *shutter*—otherwise the audience would see the film in motion as well as the still images themselves, making it much more difficult to see. In between the images flashed on the screen are moments of darkness, but thanks to persistence of vision we don't notice the darkness.

This isn't strictly true of course—you may well have noticed the flickering in the corner of your eye even if oyu don't notice it where you're focusing.

[1] Waving your hand rapidly back and forth in front of a fluorescent light can show the flicker indirectly in a strobe-like effect; near the end of a fluorescent bulb's life the frequency of the flicker can drop to levels more easily seen directly.

A small area in the centre of your retina, called the *fovea*, is what you see with in the middle of your vision. The fovea has a much denser array of light receptors, providing fine resolution; your central vision specializes in perceiving colour and shape. The rest of the retina supplies your *peripheral* vision, with a sparser layout of receptors. Peripheral vision is more sensitive instead in low light conditions and in perceiving motion—in particular, the threshold frequency at which flickering *fuses* into continuous light is much higher, meaning you're more likely to notice flicker in a light source when you're not looking directly at it.

Persistence of vision has nothing to do with the perception of motion, as is sometimes claimed, but does play an important role in animation. A fast moving light appears to us as a streak in the direction of movement, since its image from past positions persists in our vision for a short period of time. In fact, every part of a scene that moves across our eye (perhaps because our eye is moving itself) is perceived as a streak, and with all the streaks added on top of each other we end up with blur along the direction of motion: *motion blur*.

Motion blur, even though it obscures some details, is an important cue for us to understand motion. You may have seen demonstrations with strobe lights before: a strobe is an extremely brief flash of light, which in a dark room means the only light reaching your eyes will be from an instant in time, hence there is virtually no motion blur. Depending on the frequency of the flashes, a rapidly spinning fan may appear to be stopped or moving slowly in either direction—while under normal illumination it would appear to be just a blur.

Video and film cameras also are subject to an analogue of persistence of vision, resulting in images with motion blur. Generally all cameras have a shutter of some design: when it's open, light can reach the light-sensitive area (film in a traditional camera, an array of light sensors in a digital camera), and when the shutter is closed no light can get in. All light reaching the film or sensor array while the shutter is open is counted equally—

a moving light will leave a streak across the image corresponding to its trajectory while the shutter is open. Once the shutter closes, the film roll is advanced to the next unexposed frame or the sensor array is cleared, ready to take the next frame—and capture of the next frame starts from scratch when the shutter reopens. Note that this form of motion blur is then slightly different from what occurs in our visual system: for film each frame has blur from a discrete interval in time that doesn't overlap with other frames' intervals, whereas our visual system is continuous with no notion of discrete frames. When we watch a film of motion captured by a camera, instead of watching the motion "live", we do have a subtly different experience of the motion blur—quite how much this matters, and if there's something we can or should do about it, is still an open research question in computer animation.

Cameras generally provide some control over the shutter time, and hence the amount of motion blur. However, there are some constraints beyond the obvious upper bound that the shutter can't be open longer than the total time for a single frame. To get a good image, film or a sensor array needs to be adequately *exposed*, i.e. receive enough light. If it's *underexposed*, with too little light, the image will be dark and grainy/noisy—many parts of the image will receive too few photons to reliably detect the true light level. If it's *overexposed*, with too much light, the image will be saturated: the sensors will report the maximum amount they can detect (maybe pure white) and give no information about how much light above that limit there was. Depending on the sensitivity of the film or sensor (the "ISO" rating) and the amount of light in the scene itself, the camera has to compensate by allowing more or less light to reach the film/sensors for the desired exposure. This can be done by changing the size of the *aperture* (the hole through which light enters the camera) and the shutter time. We'll discuss the effect of the aperture later; the amount of motion blur present in each frame is proportional to the shutter time for obvious reasons. For artistic reasons a director may want less motion blur for a shot with fast-moving action—so that there is more detail apparent in each frame to tell the story—or pos-

sibly the director may actually want more motion blur to emphasize the motion itself.

Stop-motion photography is a classic animation method, where animators pose each individual frame (e.g. using modeling clay, which sometimes goes under the name "claymation") and take a photo before moving to the next. One of the artifacts of stop-motion is that there is no motion blur in any frame, since the scene and the camera aren't moving. This has its own instantly recognizable charm, but also makes it obvious that the motion is not real. This becomes a real issue for special effects work where stop-motion is used to animate effects that couldn't be done in real life but need to be integrated in a *live action* film—sometimes with *elements* such as actors filmed with regular cameras in the same frame as the effects. If part of the frame has motion blur but some of it doesn't, it sticks out like a sore thumb: audiences immediately see through the effect and can lose their "suspension of disbelief". One of the great milestones in film technique, *Jason and the Argonauts* (1963), featured monsters animated with stop-motion by the great Ray Harryhausen: it is a wonderful film but it's visually obvious the monsters never actually coexisted with the actors fighting them.[2]

Several possibilities exist to add convincing motion blur to stop-motion footage that must be integrated with live action footage, for example selectively blurring each frame using image processing software. The key point is that something *must* be done to bring it back. The same holds true for purely computer-generated imagery inserted into live-action films: we have to render images with simulated motion blur that matches the mo-

[2] I sometimes am met with laughter when I point out that something like skeleton warriors fighting humans is obviously fake because of the motion blur, instead of the fact it's skeleton warriors. Yes, we all know skeleton warriors don't really exist, so logically they never could be filmed in real life. However, the point is that logically knowing an element in a story could only be imaginary doesn't necessary detract from the experience at all: suspension of disbelief is our term for an audience accepting that something imaginary is consistent with and a natural part of the context of the story. Visual artifacts can break the suspension of disbelief, jarring the viewer out of the story at an instinctive level, which is rarely desired except for comic effect.

tion blur of the real footage. Even in computer animated films with no real footage at all, realistic motion blur is generally desired to help with the suspension of disbelief. We will return to this and similar issues later.

1.2 Displays and Frame Rate

An important element between the generation or capture of animation and its perception by an audience is the display: the monitor or projection screen where the images are actually shown.

For animation, probably the most important characteristic is the *frame rate*, the speed at which the frames are displayed. Film traditionally uses a frame rate of 24 frames per second (or *fps* as it is usually abbreviated) although in the very early days of silent black-and-white film lower rates, such as 18fps, were not uncommon.

Frame rate also can apply to the number of different frames being shown in a unit of time—which can be less than the display's frame rate. For example, while film has 24 distinct frames per second, it typically is projected at 48 frames per second, with each frame being shown twice—even in your foveal vision, 24 flashes per second is a slow enough flicker to be noticeable, and quite uncomfortable for the duration of a feature film. On the other hand, most traditional drawn-by-hand animations are animated "on twos", meaning each image is shown twice, i.e. the frame rate is only 12fps. Low budget, low quality animation may just be "on threes" or even "on fours", meaning 8fps and 6fps respectively. Panning shots, where the whole scene is supposed to move past smoothly, are one of the rare cases where traditional animation may go to 24fps, "on ones".

TV in North America, under the NTSC standard, runs at 29.97fps; in Europe PAL and SECAM standards run at 25fps. It's actually a bit more complicated, as these formats are *interlaced*: each frame is composed of two passes, with the odd horizontal *scanlines* of pixels (known as a *field* shown in one pass, and the even scanlines (the other field)in the next pass. Since

the two fields are shown at somewhat different moments in time—roughly 1/60th or 1/50th of a second apart—it's normal for them to also be captured from different moments in time, making the formats almost equivalent to 59.94fps or 50fps with half the vertical resolution. Interlacing was deemed necessary (given the constraints of the hardware at the time) to prevent apparent flicker similar to film projectors double flashing at 48fps.

More modern hardware is capable of *progressive scan* display, i.e. showing the scanlines in order, at high flicker-free frame rates—making interlacing obsolete. *High Definition* (HD) formats provide for higher resolution, higher frame rate, progressive modes for video.

Parenthetically, given that film is running at a different frame rate than TV (which is itself different depending on what standard you go with), not to mention interlacing, you can imagine there are all kinds of technical difficulties and potential artifacts associated with converting from one format to another! Things get even more complex when modern home theatres attempt to undo the conversion so as to provide the original higher quality 24fps film from a converted DVD...

So what frame rate is adequate? It depends. Traditional animation on twos shows that decent motion can be appreciated at 12fps or perhaps even less, at least for certain types of more stylized content. For live footage, animation that is supposed to look photo-real, or even just panning shots where the whole view is in smooth motion, 18fps was found deficient (it looks jerky) but 24fps or higher seem generally adequate for the illusion of smooth motion. However, for games where reaction time—fast user interaction—is important, higher frame rates like 60fps are considered necessary. For displays, the frequency at which flickering light are fused into a perception of continuous light provides a natural upper bound on what is useful: higher frame rates than the flicker fusion threshold (which is approximately 50fps for foveal vision) don't provide any advantage.

2 Physics

Since most animation relates to how things move in reality, it's important to study the real physics of motion.

One route to take this is to draw heuristic lessons from physics. Conservation of linear momentum, or equivalently that every force has an opposite and equal reactive force, means it's important to portray the reactions of actions—for example, if a character jumps off a diving board, the board should deflect downwards in response to the force. Newton's second law, $F = ma$, implies that motion is reasonably smooth. Ballistic trajectories in the air should follow parabolic arcs. [3] Even very stretchy materials, and especially fluid-filled materials like flesh, tend to conserve their volume when they deform meaning when they stretch along one direction they must contract along other directions. The list could go on.

Another route, for some specific types of effects, is to actually directly solve the equations of physics to generate motion: this is called *physics-based animation*. For example, the motion of water sloshing around a boat can be awfully complex and difficult to animate by hand, yet it is governed by relatively simple partial differential equations which can be (approximately) solved automatically by a computer. We will return to this general idea later on in the notes.

3 Art

Artistic considerations dominate graphics like no other branch of computer science. Even in straight-laced scientific visualization of climate data aes-

[3] Artists, even if they don't study the actual equations of the physics, will often gather together *reference footage* and scientific descriptions of a particular phenomena to study in depth when faced with a difficult animation task—e.g. exactly what shape a boat wake should be on the open ocean. This process of study of course extends beyond physics to behavioural psychology, animal locomotion, etc.

thetics plays an important role, but in animation for film or games, art is king. It therefore makes sense to take a look at some of the principles and concepts that have been codified for traditional animation.

In a famous SIGGRAPH[4] paper, John Lasseter (director of *Toy Story* and much more) introduced to graphics the classic list of principles developed at the Walt Disney Studio, and how they could be applied directly to computer animation [Las87].

Some of the principles directly involve artistic design:

Timing: Allocating an appropriate amount of time to each action to convey the right impression including the time to prepare an audience for the action itself and the time for them to react to it.

Anticipation: Making sure actions are appropriately prepared, such as seeing a character take aim before firing a gun).

Staging: Setting up a scene so that the action is as clear as possible. One commonly used criterion is that the silhouette edges of characters and objects—more specifically the highest contrast forms—should be enough to clearly indicate what's happening.

Overlapping Action: To keep a good rhythm, a continuous flow through an animation—generally desired except for dramatic effect—the next phase of an action should begin before the previous one finishes, and more generally actions should connect to one another in a natural way.

Slow In and Out: Sometimes known as *easing in and out*, this refers to how smoothly a motion begins and ends, i.e. what the rate of acceleration or deceleration is. Artistically this is an important way to communicate how heavy or light something is, how much energy there is, what the internal emotions of a character might be, etc.

[4]SIGGRAPH is the premier computer graphics conference, held annually. Many of the great techniques we will study were originally presented at SIGGRAPH.

Arcs: Perfectly linear, straight-line motion is often both boring aesthetically and unrealistic—it's too perfect. Most motion should have some curve to it.

Exaggeration: Emphasizing or accentuating the important parts of an animation is important, pushing them beyond what might be, strictly speaking, realistic. This is of course obvious in classic cartoons, but applies even in *photorealistic*[5] visual effects work where subtly exaggerating motions, lighting effects, dimensions, etc. can convey a clearer and perceptually more "real" experience. In fact, if you ever see a real live-action film or TV set, and compare the results to raw home video, you can see how much exaggeration (in the form of artificial lights, make-up, costuming tricks, etc.) is necessary just to accurately portray what we believe we perceive every day. Even scripted dialog is exaggerated in a sense, filtering out all the realistic pauses and "um"s and unfinished sentences that are a normal part of real speech, and if it weren't we'd probably be either bored or driven to distraction instead of immersed in the story.

Secondary Action: If nothing moves apart from the primary action, the rest often looks very unnatural. Perfectly still parts of a frame can be distracting—it's as if the scene is on "pause" instead of being animated at rest. Even if the background isn't supposed to be changing at all while characters move in the foreground, a standard trick in hand-drawn animation to avoid the problem is to at least redraw the background (just tracing over the original drawing) so that there is at least some sense of "animation" to every part of the image (coming from the small variations in each drawing), making it look consistent and natural with the main motion. There are also many subtle but definite secondary

[5]The term "photorealistic" is commonly used to describe graphics that aims to look identical to a photograph of reality. However, Apodaca [Apo98, AGB00] noted that directors so often want something that's a bit beyond reality—more colourful, more attractive, more dramatic, more expressive—and also get rid of the annoyances from reality—shadows falling in inconvenient places etc.—that they coined the term *photosurreal* to describe this exaggerated more-real-than-real reality.

motions that need to be present to be convincing—eyes blinking and darting, characters breathing, grass moving slightly in the breeze, etc.

Appeal: There should be a strong sense of design in every scene—even when ugly, dull, or otherwise unappealing things are portrayed, the artist should find a way to give the portrayal itself a certain visual appeal. For example, one should be cautious about symmetry and other patterns: assymetry is almost always more interesting and appealing visually; humans are also very good at recognizing (nearly) exact repetitions, rhythms, etc. and can be eaily distracted by them if they're not explicitly desired.

Some principles reflect a hand-animation approach to physical principles or other aspects of reality we have already alluded to:

Squash and Stretch: Most objects that move are not rigid, and should be shown to deform to avoid a stilted, robotic look. High speed photography reveals that these deformations are more extreme than we might intellectually expect—try looking for strobe images of a tennis ball hitting a racket for example—but that we subconsciously miss if not present in an animation, and might need to be exaggerated even further to convey the desired effect. Squash and stretch also refers more specifically to the physical property of volume conservation mentioned above: if an object is stretched on one axis it should squash along others to maintain its volume. Finally, stretch is often used in traditional animation to give an impression of motion blur (which is otherwise extremely hard to render in pen and ink, for example) which we have noted is critical for the correct perception of speed.

Anticipation: Aside from artistic considerations, anticipation of a motion is often physically necessary too, to achieve the forces involved. To prepare for a big jump, a character has to crouch a bit first.

Follow Through: Likewise, after the end of the main part of a motion there is typically continuing motion due to momentum. When a character lands from

a big jump, the knees must bend (or break!) to absorb the kinetic energy.

Overlapping Action: While important artistically, it is also simply a fact of life that actions tend to overlap. People prepare for and start executing one physical action before finishing the last.

Slow In and Out: As mentioned above, slowing in and slowing out is all about how smooth the motion is—in particular, it reflects $F = ma$ and the act that velocities should generally be smooth.

Arcs: Many motions are physically constrained to essentially circular arcs. When you rotate your forearm at your elbow, your hand moves in a curve rather than a straight line because the bones stay a fixed length.

Secondary Action: Many secondary actions follow from physics. When a character lifts a heavy object, the primary action is the lifting motion but secondary actions of muscles flexing, tendons growing taut, and other signs of exertion all need to be there.

Finally, there is also a principle reflecting the two common approaches to producing traditional animation:

Straight Ahead: In this mode, the animator makes each frame in sequence. This is the only method available to stop-motion animators, but for other types of animation is sometimes used to achieve a fresh, uncontrolled, wild nature. To some extent it also encompasses some of the semi-automatic techniques, such as physical simulation and motion capture, used in computer animation which we will discuss later on.

Pose-To-Pose: This is the mainstay of most hand-drawn animation, where particularly important poses are carefully planned and drawn first, then the *inbetween* frames are drawn to fill the time between those poses (a process sometimes called *inbetweening* or just *tweening*). This provides separate control of posing and timing, and also provides a useful division of labour. The selection of the most important poses is generally

artistic driven, but usually starts with the *extremes*: the most dramatic poses, the fullest extents of the motion, the moments of impact.

Pose-to-pose is the analoque of *keyframe* animation in the computer setting, where poses are determined at particular frames (the keyframes) and the motion between is interpolated—generally automatically by the computer. However, computer keyframe animation differs significantly in that typically it's not the full image which is being set on the keyframes and then interpolated—rather it's different underlying aspects such as the position of a character, the angle of a joint. These various aspects can be independently keyframe animated, and often are done in a top-down *layered* approach. For example, perhaps just the position in space of a character is keyframe animated until it's about right—the character moves through the scene at the correct speed, arriving at the right places at the right times—but without the smaller motions such as the action of the limbs and head. Then the limbs might be keyframe animated, until they look right. Then the detailed motions of the fingers and the head could be added, then the motion of the eyes, the breathing, the clothing, etc. are added one layer of detail after another.

4 Pipelines

The word *pipeline* gets used a lot in computer animation. You've already seen a narrow use of it in describing the real-time rendering process underlying OpenGL and similar software: mesh and texture data is transformed and combined in several stages (model-view transformation, projection, shading, rasterization, etc.) until at the end of the pipeline the final image is formed. However, most studios (both film and games) use the same kind of model at an organizational level to divide labour and ensure effective control over the end results. This sort of pipeline has co-evolved with the algorithms and software used in computer animation, thus is worth knowing about.

Each studio's pipeline is different, and it may be different for each individual project even, but generally it follows the same general scheme. I'll discuss a film pipeline in rough order of execution, but it's important to know that it's not entirely a linear process:

- some stages of the pipeline can run in parallel or in slightly different orders depending on circumstances,
- some stages of the pipeline can start working early on with provisional test data before the real data comes through (this is called *development*, and may need to be done before a studio can be confident it can actually take on a project for real, or to figure out what budget and time will be necessary),
- and there is sometimes a cycle where problems detected at one stage must be fixed in an early stage, though these interruptions are expensive thus the pipeline is generally designed to avoid this if possible.

I will also only focus on the parts that are particularly relevant to computer animation. Scriptwriting, funding, casting, soundtrack, audio recording, colour grading, editing, distribution, etc. are all terribly important of course, but largely tangential to the animation work.

4.1 Storyboards and Concepts

The earliest input to the computer graphics pipeline are *storyboards*, a sequence of simple sketches (typically on small cards that can be pinned to a board for organization) of what important frames of the final film should look like, at least one for every shot, every important event. This allows for problems to be fixed at a point where it's just a few seconds work with pencil and eraser or a matter of shuffling a few cards around. (In some sense the whole workflow is optimized for fixing mistakes as early and cheaply as possible: one important feature of any production are *dailies*, a daily show-

and-tell of any works-in-progress from any stage of the pipeline, where supervisors or even the director can instantly give feedback on what should change before it's locked down for further stages of the pipeline.)

When scanned in and played back at the appropriate timing with a rough audio track laid down, this might be called an *animatic*, which gives a very good idea of what the final film will be like.

The storyboards are not keyframes: they aren't trying to capture every aspect of every motion. There may even be only a single storyboard image for a shot if that's all that's needed to communicate to artists further along the pipeline what they need to do.

Artists also work on *concept art* early on, often using traditional media like oil paints or clay, to visually explore what characters or settings could end up looking like. What ends up approved can be very useful reference material for further stages—for example, scultures may even be scanned in to the computer to help modelers build the final digital version.

4.2 Previz

Previsualization, or previz for short, is the art of making a rough draft of the whole film, very quickly and cheaply, with a condensed version of the pipeline. The goal is to refine the storyboards much more—pencil sketches can only get you so far—while cutting as many corners as possible to concentrate purely on the elements of interest: yet more problems can be caught and fixed cheaply in previz.

Exactly what a previz shot looks like depends on what needs to be analyzed. If it's just a matter of getting the camera angle right for a set, the set might be a fairly abstract mesh rendered in matte gray but with the right dimensions, without bothering to include textures, characters, etc. If it's the timing and placement of an explosion, it will have to be there of course, but it might be hacked in as a video of an explosion projected on a *billboard*

(a.k.a. *sprite*), a 3D rectangle with an image meant to give the illusion of more complex geometry (old video games such as DOOM used billboards for monsters, for example), stretched and slowed down or sped up to fit the scene.

4.3 Plate Work

In a visual effects show, as opposed to a purely computer-generated animation, there is an added complication that the digital effects often need to be applied to real footage that has already been shot, called *plates*. This is probably where the biggest use of computer vision techniques comes into play.

Some effects are purely image processing and don't really involve the 3D parts of the pipeline, for example painting out the wires holding up a stuntman in a harness so it looks like they're flying through the air on their own.

One image processing task we will look at in more depth is creating a *matte*, essentially a labeling of every pixel in an image as to what we want to keep and what should be thrown away. For example, an actor might be filmed in front of a *green screen* (or some other colour background that is easy to distinguish) and the matte would select only the parts of the image that are the actor, not the background. The matted image of the actor can then be placed atop a digitally-created background. In fact, the label is usually going to be a factional value, called *alpha*, which goes from zero (meaning do not keep this pixel, i.e. make it fully transparent) to one (keep this pixel as is, i.e. make it fully opaque) with intermediate values indicating a pixel that is only partially kept—typically at the edges of an object.

Other effects involve 3D computation. To add a 3D digital effect to a plate, or a filmed actor to a 3D digital image, it's critical that the virtual camera used for 3D rendering matches up to the real camera—both in terms of its intrinsic parameters like aspect ratio and field-of-view and its spatial relationship to the scene (where it is and in what direction it's pointing). This

is especially true, and especially tricky, for moving cameras. This process is called *matchmove*, and is typically done via computer vision techniques: identifying special markers in the plate (which might be painted over later for the final images) and computing the camera parameters based on where they appear in the image.

Other related tasks may not use the actual plates but other photos or measurements taken on set. For example, it is standard to take a panaromic high dynamic range (HDR) image to capture the surroundings of the set, and in particular the lighting—this can be used directly to compute the simulated lighting of 3D models which are to be added.

4.4 Modeling and Rigging

A geometric model must be designed for just about every digital character, object and setting in the film.[6] There are a multitude of different techniques for modeling, most of which we won't cover in this course, but most revolve around editing polygon meshes. Modelers generally prefer to work with *quad meshes*, meshes whose faces are quadrilaterals; operations include deformation (moving vertices of the mesh around), subdivision or refinement (splitting a large quad up into four smaller quads), extrusion (growing a block out of a patch on the surface), and many more.

Modelers are also increasingly using 3D scanners. Rather than just use reference sculptures to look at or measure, there are now readily available scanners which can automatically construct an accurate 3D model of most objects.[7] The models produced from scanners generally aren't directly usable—the mesh tends to be a dense unstructured mess—but can be used as a very convenient reference directly within a modeling program, where the artist produces a much cleaner, structured, sparse mesh.

[6] Occasionally other techniques like billboards can be used to get away without modeling.

[7] Some objects, such as those with mirror or transparent surfaces, are still troublesome to scan.

A well-behaved mesh makes the job of setting up well-behaved texture coordinates much easier, which is critical for later stages.

Models of objects that move or deform in some way, characters in particular, need to be *rigged* as well. A rig is, in essence, a user interface for controlling the deformation of a model—a parameterization of the ways it can be changed. The rig is what animators will later use to pose models, and what the computer will use to interpolate between keyframes.

4.5 Texturing

A model on its own is abstract geometry: to be rendered it needs a *shader*. A shader is a specification of what colour is produced at any point on the model as a function of incoming light and the outgoing view direction. Most often shaders are written in a special *shading language*, such as Renderman Shading Language (the language associated with RenderMan, probably the most commonly used renderer in film), GLSL (the language associated with OpenGL), or Open Shading Language (a language designed for modern advanced raytracers in particular). Shaders can be very complex programs—tens or even hundreds of thousands of lines of code—and shader development itself is an important job.

Some shaders can generate surface details *procedurally*, i.e. as straight mathematical functions of position / texture coordinates / normals / etc. For example, a diffuse grey surface where the grey level is $k(1 + \cos(u)) * (1 + \cos(v))$ (as a function of texture coordintes (u, v)) would have a regular smooth array of alternating black and lighter grey. One of the most common building blocks for such shaders is *noise*, a class of pseudo-random but not periodic function first developed by Perlin [Per85] and since elaborated in many ways.

However, most shaders achieve surface variation through texture inputs. Part of the job of texturing is to also produce those texture images for models. These might be painted either as 2D images or directly on a 3D model.

4.6 Layout

Layout is basically putting models together into sets—furnishing rooms etc.

4.7 Animation

Animation takes the rigged models, the prepared sets, the previz version of the shot, and creates the primary motions—especially characters.

The classic way of doing this is, as mentioned before, keyframe animation. Artists use the rigs to pose models at selected times and guide the computer in interpolating the motion between. Also as mentioned previously, this is often done in stages. *Blocking* is the first step: putting the characters in the right places in the right times, without worrying about animating their limbs or anything else—kind of like moving dolls around. After that has been properly tuned, the component parts of the characters are keyframe animated (probably using a different selection of frames).

An alternative that's gaining in popularity for certain types of shots is to instead use *motion capture*, or *mocap* for short. Here a real actor is filmed acting out the motions, and computer vision algorithms are used to reconstruct the 3D motion to be mapped to the rigged model.

Looking into the future, more automatic methods may start to be used as well. Research has advanced significantly in getting the computer to realistically animate humans walking and doing other tasks under high-level guidance from users. This sort of approach has already been widely adopted for simulating crowds where individually animating every one of ten thousand people would be impractical.

4.8 Effects

The animation stage of the pipeline is analogous to acting: creating the primary motions of the characters, that express emotions, etc. There is a lot more motion to handle, though, as mentioned above under Secondary Action. This generally falls under the category of effects. For a character this could include the motion of the flesh in response to the primary action, the hair, and the clothing. Away from characters there's grass, water, smoke, fire, dust, explosions, or more. Procedural methods and, increasingly, physical simulations play a big role in effects.

4.9 Lighting

When shaders operate inside the renderer, one of the most important inputs is lighting information for a scene. Lighters are responsible for arranging this, along with the shadows they might cast.

4.10 Rendering

Once all of the above is set up, the final renders can run. Even for a single frame, many separate images may be produced (in one or more passes) to preserve some control over the final result as late as possible in the process. Rendering in 3D at the quality demanded for film is still expensive, taking hours per frame[8], so last-minute decisions about the look that require a re-render may be impossible. If the foreground has been rendered separately from the background, and the shadows separately from the highlights, etc. then the balance between these can be adjusted with cheap image techniques.

[8]Even as computing power and algorithms have improved, audience expectations and artistic ambitions have added more complexity, so rendering times haven't changed much. The continuing progress in GPU power may potentially change this.

4.11 Compositing

After rendering is complete, the various images that contribute to a single frame of the film (both computer-generated and plates) can be combined in an operation called *compositing*.

At its simplest, compositing computes each output pixel as some function of the input pixels at the same location—for example, adding their values together, maybe first multiplying by an alpha channel (a matte) or applying colour adjustments. More complicated operations are common as well, such as blurring to controllably simulate motion-blur or depth-of-field.

4.12 Outside the Pipeline

I should also point out there is a lot more technical challenges underpinning this pipeline in serious computer animation work. An effects-heavy film these days can end up using over a petabyte of data across millions of files; the challenges of storing this, backing it up, making it secure, ensuring high speed access, and keeping it all organized despite hundreds of people using and modifying it, are formidable. A big studio may also have tens of thousands of computers in their *render farm* (cluster of servers for sending tasks such as rendering to), on top of all the artist workstations—networking all of this, figuring out power and cooling, and just keeping it all running is also a big deal.

5 Exercises

5.1 Reading and Watching

For more on the human visual system, which is a huge and fascinating subject of research, Palmer's text [Pal99] is a great start. For the sake of this

course, you don't need to know anything more than covered here, but the importance of understanding human perception will only increase in the field going forward.

Find and watch *Jason and the Argonauts* (at least the skeleton warrior sequence) or other classic milestones in animation.

Watch some of a film (perhaps an animated movie, but anything is OK) without the sound and at half speed or slower; single step through frames when interesting motion is happening. Notice how obvious the motion blur is.

Many DVDs of effects-heavy films or computer animated films have extra features showing how some of the scenes were put together. Find and watch one. For example, *Moon* (2009) has a great description on the DVD of how they did many of the effects—and it's also one of the best science-fiction films ever made, so definitely worth watching.

Find and watch high speed photography of everyday motion like a tennis ball hitting a racket.

5.2 Programming

In this course we will primarily be using Python 2.7 (though earlier versions may work). For performance reasons (and to some extent sheer inertia) most animation softwaer is actually written in C++ these days, especially for games, but Python has become the de facto standard for scripting in the film world and is an excellent general purpose language to be able to use. We will not need to scale up to massive data sets or high performance, but do need a language that makes it easy to rapidly develop prototypes and experiment with—which makes Python a superior choice.

For GUI work, we will rely on the Qt library, and in particular the PyQt4 bindings for Python (though the alternative PySide bindings should also

be usable with minimal porting effort). Qt has emerged as the best cross-platform GUI library available now, and probably the best library for Linux development in particular—certainly judging by its uptake in the animation software industry.

We will also use OpenGL for rendering, with the PyOpenGL bindings, and of course using the GLSL shading language with it. While RenderMan is the standard choice in film, and is unarguably a far more elegant API from the standpoint of high quality "offline"[9] rendering, modern OpenGL is rapidly catching up in terms of capabilities, and is a natural choice for real-time work (particularly going cross-platform, or on mobile devices with the dominant OpenGL ES standard). Raytracers are also gaining in popularity for film, but haven't yet made much of a dent in the real-time space, so OpenGL it is.

The first task, then, is to figure out how to install Python 2.7, PyQt, and PyOpenGL if they're not already available.

If you haven't programmed in Python before, start getting up to speed: the official tutorial available on the `python.org` website is an excellent start, but there are also many other tutorials and books available. Be cautious, however, to stick with Python 2.x documentation rather than Python 3.x: the newer Python 3 series introduced some incompatibilities when it upgraded the language. Currently there are still some issues with common libraries not supporting Python 3, which is why we haven't jumped to it yet.

The Qt library is fairly large, and has its own unique notion of "slots" and "signals" to connect objects together. Qt itself is a slightly-enhanced C++ project, so reading standard Qt tutorials doesn't perfectly translate to PyQt. However, there are some specific PyQt tutorials you can find to get your feet yet. An excellent exercise specifically for this chapter is to translate from C++ to Python the standard C++ "scribble" Qt example, found at

[9] Offline means non-interactive: you submit a rendering job and at some point later get back the final image.

http://doc.qt.nokia.com/latest/widgets-scribble.html

Probably the single most tricky issue is that Python has less explicit control about argument passing (whether by reference or value) than C++. In particular, the following C++ code,

```
void ScribbleArea::resizeImage(QImage *image,
                               const QSize &newSize)
{
    ...
    QImage newImage(newSize, QImage::Format_RGB32);
    ...
    *image = newImage;
}
```

does **not** translate to the following Python code:

```
def resizeImage(self, image, newSize):
    ...
    newImage = QImage(newSize, QImage.Format_RGB32)
    ...
    image = newImage
```

This Python code will create the `newImage` object, but instead of assigning it to the caller's `image`, it will simply reassign the method's local `image` name to refer to `newImage`: the caller's version remains unchanged. Instead, change the method to return the new image:

```
def resizeImage(self, image, newSize):
    ...
    newImage = QImage(newSize, QImage.Format_RGB32)
    ...
    return newImage
```

and a caller can do something like

```
visibleImage = self.resizeImage(self.image, self.size())
```

in lieu of the C++ statements

```
QImage visibleImage = image;
resizeImage(&visibleImage, size());
```

Note that I am assuming you use

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
```

at the start of the Python files, to get the PyQt classes in scope. With this exercise you should be able to get a functioning albeit primitive drawing program running.

Once that's done, the real work for the chapter is to extend this to a "flip-book" animation program, that stores a hundred or so frames internally and lets you draw on them (in black and white) and play them back. As an extra useful effect, try showing through the frame before and after in light grey to be able to better judge how the motion will look. Use your program to animate some simple scenarios, like a bouncing ball or a stick figure walking, and then get creative.

Sample code to get you started on FlipBook will come online soon, along with more precise instructions on what to do and what to hand in.

5.3 Sample Exam Questions

This chapter has no algorithms or interesting math, but plenty of terms to know (just about any word or phrase in italics) and several facts about

practical aspects of animation with underlying explanations you should be able to discuss. Possible exam questions include:

- How is persistence of vision related to motion perception? (Note: this isn't a straightforward question!)
- Why is the frame rate for film different from the rate at which it's projected?
- Discuss two physical principles covered by Squash and Stretch.

In general you can expect exam questions of this nature, to complement the practical work in assignments.

References

[AGB00] A.A. Apodaca, L. Gritz, and R. Barzel. *Advanced RenderMan: creating CGI for motion pictures*. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 2000.

[Apo98] Anthony A. Apodaca. Photosurrealism. In *Proc. Eurographics Rendering Workshop*, pages 315–322, 1998.

[Las87] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *Proc. SIGGRAPH*, pages 35–44. ACM, 1987.

[Pal99] Stephen E. Palmer. *Vision science : photons to phenomenology*. MIT Press, 1999.

[Per85] Ken Perlin. An image synthesizer. In *Proc. SIGGRAPH*, pages 287–296. ACM, 1985.