DANM 220 2012

Topics for DANM 220 Winter 2012.

Date	topic	Objectives	demo	assignment
Jan 9	Essential Background	Coordinates, colors, Define frame structure	Setup	Set up Max
Jan 11	Basic drawing	Code structure, functions,	Rorschach	
Jan 16	Holiday			
Jan 18	Animated drawings	The draw loop	Bouncer	Random drawing
Jan 23	Movie munging	pixel color, color processing	Load movie, Canned filters,	
Jan 25	repos	Image distortion	ReposTut	Filter a movie
Jan 30	Drawing with trigonometry	Function based images	Arabesques	
Feb 1	Feedback tricks	Iterative processing, fractals	Feedback, ferns	triangle
Feb 6	Live video	Video capture & analysis	Zen mirror,	
Feb 8	object tracking	Trigger events from motion	hotspots	Follow the post-it
Feb 13	The 3D world	Intro to openGL	GLtut	
Feb 15	More 3D		Bouncer3D	
Feb 20	Holiday			
Feb 22	Sound &visualization	Playback, capture, Audio to image	Simple loop, lissajou, waveform	visualizer
Feb 27	shaders	The GPU	Rota, chladni, mandelbrot	Work on projects
Feb 29	gen	Low level coding		""
Mar 5	Input	Keyboard, mouse & external devices	HI,MIDI,OSC, firmata	""
Mar 7	Output	External control	DMX, firmata	""
Mar 12	Crit	Student projects		
Mar 14	Wrapup			

Assignments:

- Random Drawing: create an image with some aspects based on chance.
- Filter a Movie: make a movie unrecognizable but interesting
- Triangle: create a program that builds layers of Sierpenski triangles
- Follow post-it: use findbounds to trace an object's motion.
- Visualizer: write a patch that responds to music.

What exactly can a computer do for the artist?

- Projections and screen display.
- Control artistic contraptions.
- Change the audience experience with responsive systems.
- Surprise us.
- Evolve.

The realities of the digital canvas.

The digital canvas is some sort of computer screen or a projection. The colors you get will be somewhat different from model to model. (Imagine what kind of painting you'd get if the color of the canvas kept changing.) Many projectors have a pixel architecture that is different from computer screens, say 1280 X 720. Converting a 1024X 768 image will require rescaling, which often introduces lateral lines or other distortion.

The level of detail is limited by the screen resolution. It is also limited by the power of the computer hosting your application. There are millions of pixels in a single frame of video, and calculating a frame will take many machine cycles per pixel. All of a sudden a billion operations per second does not seem very fast. Computers have both a CPU (central processing unit) and GPU (graphics processing unit, generally part of the display card.) The GPU is about a thousand times faster than the CPU, but it is often awkward to use. For instance, GPU programming is done in a different language. (We can program the GPU in Max, but not in processing.) In all cases there is a tradeoff between resolution and time.

The time/resolution tradeoff means the frame rate is often less than the optimum 30 fps. We usually don't notice until the rate goes below 20 fps, but our perception change is sudden. This happens at different rates for different people. Slow rates may be tolerated for images that don't change much.

The third dimension.

The world has become accustomed to very real 3D images from features like "Finding Nemo" and "Avatar". Of course those are produced on render farms with 1000 machines producing a few minutes a day. A more realistic goal is game level realism. That means simple textures, less subtlety in lighting, and so on.

We specify 3D images through a language called openGL. Both Max and processing support openGL, and try to help with the more tedious aspects of working in that language. Max 6 offers important improvements in game style graphics.

Image and sound.

Our works of art need not be silent. Processing will allow cued playback of recorded material and samples, as well as some elementary synthesis. Max/MSP is a full fledged

synthesis and DSP environment. The two can coexist and communicate, so using Processing for images and Max for sound is perfectly feasible. Either can also communicate with most audio platforms such as Reactor, Supercollider, ChucK and Csound. Max can host your choice of plug-ins.

Creating images out of sounds is also feasible in either language.

Transformation of pixels

A lot of video art is based on transforming existing video footage. At one time there were modular video synthesizers to complement the more famous Moogs and Buchlas. These performed analog processes like threshold switching and slew limiting. In digital processing, it is the color of pixels that is modified. The modifications can be based on several things. For instance, if there are two video sources, the color of a pixel may be compared to a specified color. If they match, the output is taken from the second source. Otherwise the first source is passed through. Thus everything colored green in the A signal is replaced by colors from the B signal.

Algorithmic drawing

Tools like Illustrator let us use the computer as a canvas, transferring ideas from our head to the screen. Max and Processing move some of the decisions to the computer. All we have to do is invent rules for the computer to follow. The rules are necessarily based on mathematics, but the math need be no more complex than what's needed to balance a checkbook. One beauty of algorithmic drawing is the rules are controlled by parameters supplied by the artist. This means many images can be made from the same algorithm.

- *Geometry* produces straight line figures. You can do a lot with straight lines, especially if they make up tiny triangles. The underlying concept is the vector, but that is only an instruction of where to go.
- *Trigonometry* produces curves. These curves are defined by angles and functions based on the angles. These functions are deep math, but the computer will perform them for us.
- *Iterative function systems* apply functions to input, then apply the same function to the results. When we get an image we like, we quit. This kind of thing can produce remarkable images including plants, mountain ranges and snowflakes.
- *Chance operations* inject random functions into a process so each run produces a different image.

Dynamic Drawing

Computer art does not have to stand still. If we use time as parameter the image can be redrawn with subtle changes or drawn fast enough that the eye is fooled into perceiving motion.

- *Loops* perform the same pattern of operations over and over again. This gives simple animation, or a painting that follows the time of day.

- *Physical models* follow some of the rules of the real world. This produces action like bouncing balls or growing vines. This also includes chaotic systems with repeatable but surprising behavior.
- *P-systems* take modeling down to molecular level. They can be used to create fire and smoke and flocking birds.

Noticing the world

Computer programmers often speak of real time. This is code that runs fast enough that response seems simultaneous with input. Even if there is noticeable delay, an art program can respond to changing environment or requests by users (whom we define as someone who is operating the program, but did not write it). This input takes two forms:

- *Camera input* captures images. These images can be used as source material for image processing or analyzed to provide control parameters for the program.
- *Physical sensors* can detect practically anything. We can build a system that is played like a violin or tracks the humidity over several days.
- *Network links* expand the input range to include a large part of the world. Again the foreign data can provide raw material or control.

Controlling the world

The program output is not limited to images on a screen. Anything that is electrical can be controlled, either directly from an Arduino style circuit or a standardized network protocol. A program can generate instructions to a performer or mechanic or another computer program.

The way of code

The only true mastery of a computer comes through the ability to write code. When we use canned software, we are bound to a certain world view, accepting the priorities and paradigms of the software's author. With millions of programs available, it seems we should be able to find something that matches our needs and methods, but we soon realize the search takes longer than the task, and we settle for anything that will get the job done.

It is absurd to consider writing any application from the ground up. Fully formed applications require complex operations for screen display, file storage and retrieval, communications and numerous other functions. These problems have been solved again and again-- tackling them is a waste of our time. Instead, we build on other's work, adding our code to a platform of routines that satisfy the standard needs. We will work with one of two platforms:

Max/MSP/Jitter lets us assemble complex programs from building blocks of working code. The process seems something like Legos at first. As with Lego building, you are limited to the objects already provided, and these objects force a certain overall shape. However, there are so many objects available that just about anything can be done within

the system. Unlike Lego, it is possible to invent new objects. There are enough people doing this now that Max has reached a critical mass, and the possibilities are expanding faster than any one person can reach the limits.

Processing provides a nearly complete application called a sketch. It is your job to complete the application by writing two or three unfinished sections of code. The result is a Java applet, suitable for adding to a web site or running an installed art work. The scope of Processing is not limited to computation and screen display. It can generate audio and MIDI or communicate with Arduino to control external hardware.

In either platform the development process goes through the same steps, steps necessary to many complex undertakings. The steps are specification, design, coding and testing.

Specification

An application begins with a problem to be solved or an operation to be performed. The more precisely the problem or operation can be defined, the easier the code will be to write. It is possible to change your mind, to add features or remove them, but the later this happens the more wasted work. Changing specs midway can also lead to internal inconsistencies that will wreck the entire program. Specification is not as formal or complex as it sounds. It's just a list of inputs and outputs with a description of how they relate. If you think of it an instruction manual, it becomes a much less academic undertaking.

Design

Design is the creative phase, the invention of algorithms to get the work done. A lot of these algorithms have already been invented, so your first task is to research the field and find them, or something close enough to be modified. Resist the urge to write code during the design phase, except little experiments to test something. The result of the design phase is a flow chart showing how one action leads to another. Make sure every input and output in the specification is included.

Coding

Writing code is the tedious part of the process. If you work in Max, the code resembles the flow chart. In Processing, every object in the chart is represented by a paragraph of code that defines a function and the structure is defined by calling functions in the draw routine. Coding is made difficult by the terse vocabulary and rigid syntax of Processing or the subtle gotchas and annoying inconsistencies of Max.

Testing

Testing actually begins as soon as there is enough code to test. If the flow chart is complete, the input and output of the functions will be clear enough to try each out as they are written. Once the whole thing is operational, give it to someone else. It will immediately break. Find out how your tester broke it and figure out the problem. Repeat the process until your tester cannot break it. It is tempting to test it yourself, but you will be amazed at how your intimate knowledge allows you to subconsciously avoid dangerous operations.

Revision

Testing and use will inevitably lead to revisions of the program. When you do this, don't just jump in and modify code. Go back and change the specifications, then see how wide ranging the effects of the change will be.

Documentation

The initial specification can be revised into instructions for using the program. You should also include comments in your code that describe the operations of each section. Make these clear enough for a stranger to understand[1].

[1] That stranger is you, in about 6 months.