

C++ is fun – Part 17

at Turbine/Warner Bros.!

Russell Hanson

Syllabus

- 1) First program and introduction to data types and control structures with applications for games learning how to use the programming environment Mar 25-27
- 2) Objects, encapsulation, abstract data types, data protection and scope April 1-3
- 3) Basic data structures and how to use them, opening files and performing operations on files – April 8-10
- 4) Algorithms on data structures, algorithms for specific tasks, simple AI and planning type algorithms, game AI algorithms April 15-17
- Project 1 Due – April 17
- 5) More AI: search, heuristics, optimization, decision trees, supervised/unsupervised learning – April 22-24
- 6) Game API and/or event-oriented programming, model view controller, map reduce filter – April 29, May 1
- 7) Basic threads models and some simple databases SQLite May 6-8
- 8) Graphics programming, shaders, textures, 3D models and rotations May 13-15
- Project 2 Due May 15
- 9) Threads, Atomic, and Exceptions, more May 20
- 10) Gesture recognition & depth controllers like the Microsoft Kinect, Network Programming & TCP/IP, OSC May 27
- 11) Selected Topics June 3
- 12) Working on student projects - June 10
- Final project presentations Project 3/Final Project Due June 10

Cinder, Open Frameworks, and OSC!

FEATURED PROJECT

PLANETARY

ADDED ON MAY 16, 2012

by [Bloom](#)

Planetary is a stunningly beautiful way to explore your music collection, available on iPad. Fly through a 3D universe dynamically created by information about the recording artists you love.

[VIEW >](#)



FEATURED PROJECT

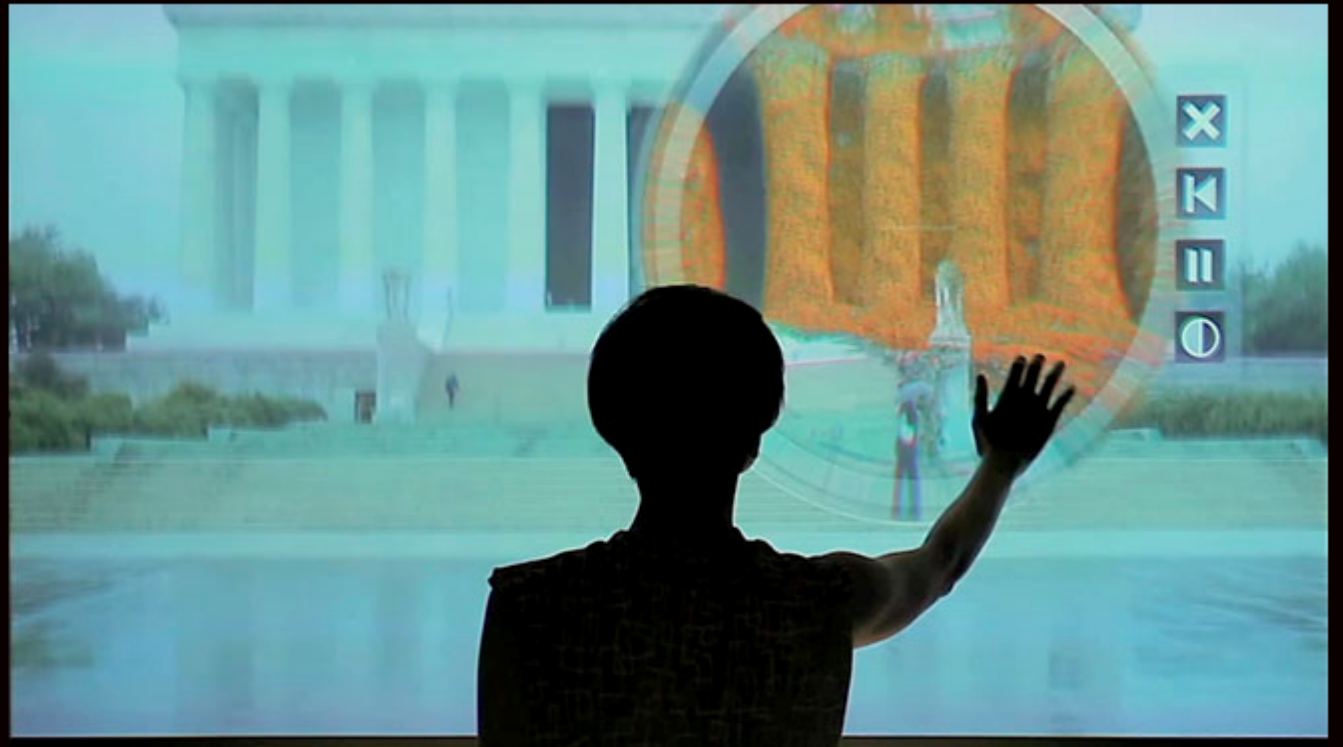
MILL TOUCH

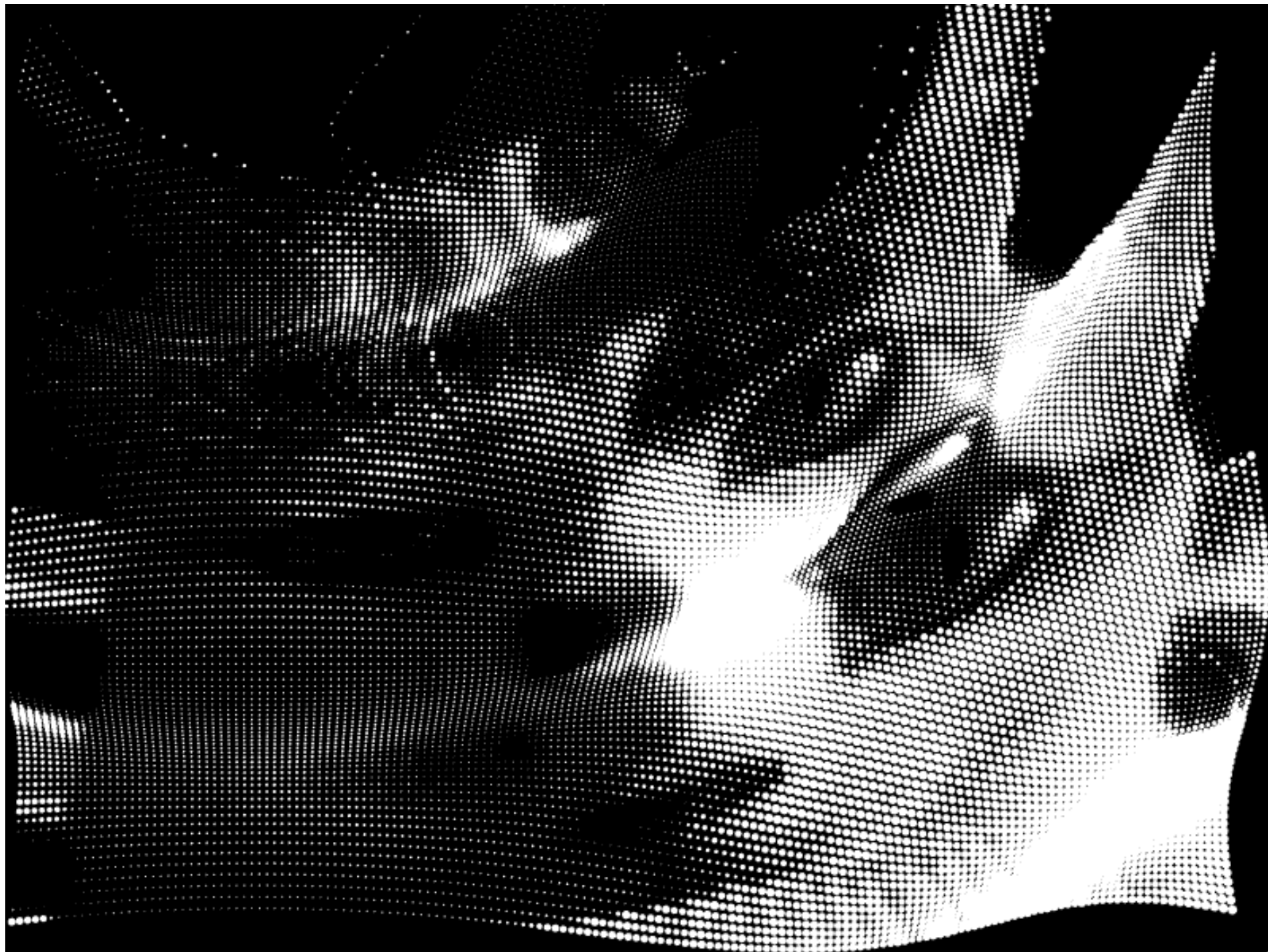
ADDED ON MAY 16, 2012

by [The Mill](#)

A rear projected, 5'x 3' multi-touch interactive screen made entirely of switchable glass featuring the full history and portfolio of The Mill.

[VIEW >](#)





What was that?! ^^

```
#include "cinder/app/AppBasic.h"
#include "cinder/Rand.h"
#include "cinder/Vector.h"
#include "ParticleController.h"
```

```
using namespace ci;
using std::list;
```

```
ParticleController::ParticleController(){}
```

```
ParticleController::ParticleController( int res )
```

```
{
    mXRes = app::getWindowWidth()/res;
    mYRes = app::getWindowHeight()/res;

    for( int y=0; y<mYRes; y++ ){
        for( int x=0; x<mXRes; x++ ){
            addParticle( x, y, res );
        }
    }
}
```

```
void ParticleController::update( const Channel32f &channel, const Vec2i &mouseLoc )
```

```
{
    for( list<Particle>::iterator p = mParticles.begin(); p != mParticles.end(); ++p ){
        p->update( channel, mouseLoc );
    }
}
```

```
void ParticleController::draw()
```

```
{
    for( list<Particle>::iterator p = mParticles.begin(); p != mParticles.end(); ++p ){
        p->draw();
    }
}
```

Source code in:

“17. Cinder, OSC, Open Frameworks/
Ch 3 Particle Tutorial src”

```

#include "Particle.h"
#include "cinder/Rand.h"
#include "cinder/gl/gl.h"
#include "cinder/app/AppBasic.h"

using namespace ci;

Particle::Particle()
{
}

Particle::Particle( Vec2f loc )
{
    mLoc                = loc;
    mDir                = Rand::randVec2f();
    mDirToCursor        = Vec2f::zero();
    mVel                = 0.0f;
    mRadius             = 0.0f;
    mScale              = 3.0f;
}

void Particle::update( const Channel32f &channel, const Vec2i &mouseLoc )
{
    mDirToCursor        = mouseLoc - mLoc;

    float distToCursor = mDirToCursor.length();
    float time         = app::getElapsedSeconds();
    float dist         = distToCursor * 0.025f;
    float sinOffset    = sin( dist - time ) + 1.0f;

    mDirToCursor.normalize();
    mDirToCursor        *= sinOffset * 100.0f;

    Vec2f newLoc        = mLoc + mDirToCursor;
    newLoc.x            = constrain( newLoc.x, 0.0f, channel.getWidth() - 1.0f );
    newLoc.y            = constrain( newLoc.y, 0.0f, channel.getHeight() - 1.0f );

    mRadius             = channel.getValue( newLoc ) * mScale;
}

void Particle::draw()
{
    //gl::color( Color( 1.0f, 1.0f, 1.0f ) );
    //gl::drawVector( Vec3f( mLoc, 0.0f ), Vec3f( mLoc + mDirToCursor * 15.0f, 0.0f ), 6.0f, 3.0f );
    gl::drawSolidCircle( mLoc + mDirToCursor * 0.2f, mRadius );
}

```



```

#include "cinder/app/AppBasic.h"
#include "cinder/ImageIO.h"
#include "cinder/gl/Texture.h"
#include "cinder/Channel.h"
#include "cinder/Vector.h"
#include "ParticleController.h"

// RESOLUTION refers to the number of pixels
// between neighboring particles. If you increase
// RESOLUTION to 10, there will be 1/4th as many particles.
// Setting RESOLUTION to 1 will create 1 particle for
// every pixel in the app window.
#define RESOLUTION 5

using namespace ci;
using namespace ci::app;

class TutorialApp : public AppBasic {
public:
    void prepareSettings( Settings *settings );
    void keyDown( KeyEvent event );
    void mouseMove( MouseEvent event );
    void mouseDrag( MouseEvent event );
    void setup();
    void update();
    void draw();

    Channel32f mChannel;
    gl::Texture mTexture;

    Vec2i mMouseLoc;

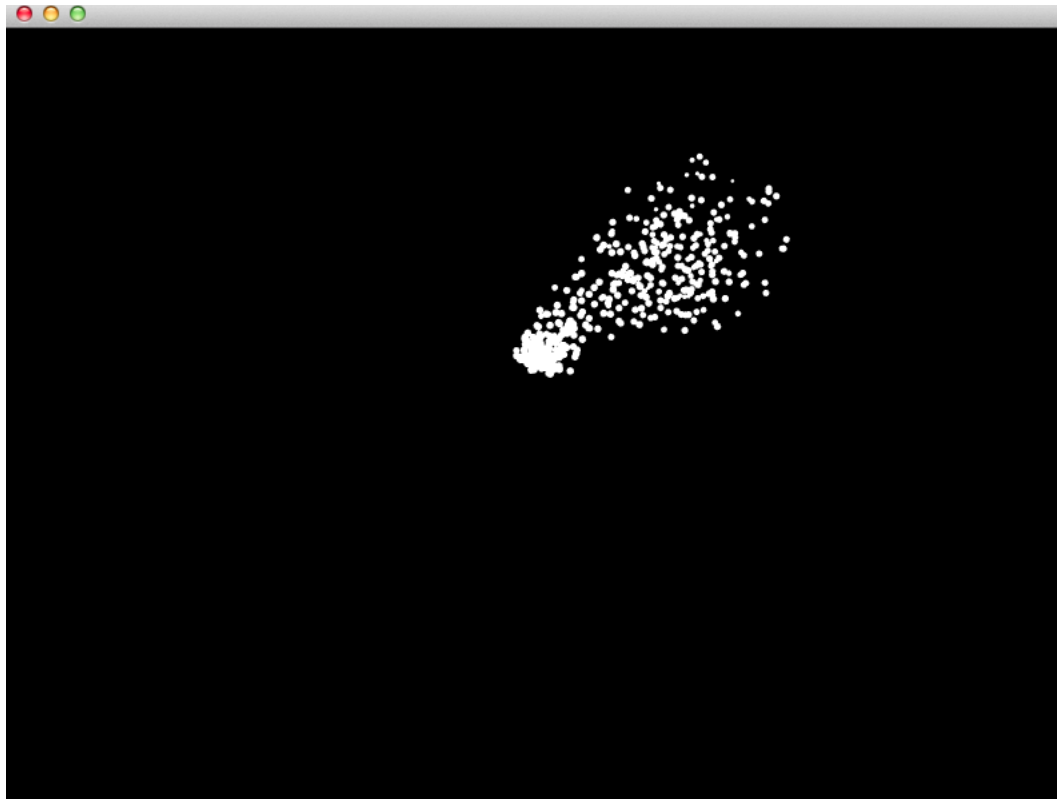
    ParticleController mParticleController;

    bool mDrawParticles;
    bool mDrawImage;
};

void TutorialApp::prepareSettings( Settings *settings )
{
    settings->setWindowSize( 800, 600 );
}

```

Source code in:
"17. Cinder, OSC, Open Frameworks/
Ch 4 src"



CREATING PARTICLES WITH MOUSE EVENTS

We are going to need to beef up our mouse related code. First up, we will add `mouseDown()` and `mouseUp()` methods to our project. We will also make a boolean that will keep track of whether a mouse button is pressed.

```
void mouseDown( MouseEvent event );
void mouseUp( MouseEvent event );
bool mIsPressed;
```

If any mouse button is pressed, `mouseDown()` will fire. Inside that function, all we do is set `mIsPressed` to `true`. If `mouseUp()` is called, `mIsPressed` will be set to `false`. Easy enough.

```
void TutorialApp::mouseDown( MouseEvent event ) {
    mIsPressed = true;
}
void TutorialApp::mouseUp( MouseEvent event ) {
    mIsPressed = false;
}
```

Finally, in our `App` class `update()` method, we add an `if` statement that checks to see if `mIsPressed` is true. If it is, then have the `ParticleController` make some new `Particles`.

```
if( mIsPressed )
    mParticleController.addParticles( 5, mMouseLoc );
```

We have gone ahead and changed the `addParticles()` method in `ParticleController` to take both the number of `Particles` we want as well as the location where we want to initially put them.

You might be thinking, "Hey, wait. If we make 5 particles and place all of them at the location of the cursor, we will only see 1 particle." We remedy this situation by adding a random vector to the location when we create the new `Particle`.

```
void ParticleController::addParticles( int amt, const Vec2i &mouseLoc ) {
    for( int i=0; i<amt; i++ ) {
        Vec2f randVec = Rand::randVec2f() * 10.0f;
        mParticles.push_back( Particle( mouseLoc + randVec ) );
    }
}
```

PARTICLE DEATH

If we allow every `Particle` to live forever, we will very quickly start dropping frame rate as hundreds of thousands of `Particles` begin to accumulate. We need to kill off `Particles` every now and then. Or more accurately, we need to allow `Particles` to say when they are ready to die. We do this by keeping track of a `Particle`'s age.

Every `Particle` is born with an age of 0. Every frame it is alive, it adds 1 to its age. If the age is ever greater than the life expectancy, then the `Particle` dies and we get rid of it. First, lets add the appropriate variables to our `Particle` class. We need an age, a lifespan, and a boolean that is set to true if the age ever exceeds the lifespan.

```
int mAge;
int mLifespan;
bool mIsDead;
```

Be sure to initialize `mAge` to 0 and `mLifespan` to a number that makes sense for your project. We are going to allow every `Particle` to live until the age of 200. In our `Particle`'s `update()` method, we increment the age and then compare it to the lifespan.

```
mAge++;
if( mAge > mLifespan )
mIsDead = true;
```

Just having a `Particle` say "Im dead" is not quite enough. We need to also have the `ParticleController` clean up after the dead and remove them from the list of `Particles`. If you look back at the `ParticleController` `update()` method, you see we are already iterating through the full list of `Particles`. We can put our death-check there.

```
for( list<Particle>::iterator p = mParticles.begin(); p != mParticles.end(); ){
if( p->mIsDead ) {
p = mParticles.erase( p );
}
else {
p->update( channel, mouseLoc );
++p;
}
}
```

Want a job doing C++? Better know some keywords and be able to talk about them

Job Title: C++ Application Integration

Job Description:

The Job Description is as follows:

-->Application Integration(L3)-

-->Batch Processing(L3)-Candidate skills include: Design of batch scripts and jobs Analysis of batch scripts and jobs Monitoring and Trouble shooting

-->**Unix Application Programming(L4)**-Networking Protocols, Advanced signal handling(concept of signal masks,signal sets, POSIX signal handling, race conditions etc), Timers, resource limits, Unix debugging and tracing features Advanced unix user space commands (archival/restore, SCM, cron etc) Concept of threads, multithreading, advanced i/o, file and record locking Advanced system design, Knowledge of unix variants, porting considerations etc Performance tuning and optimization

-->**C++(L2)**-Should be working in a C++ Project. Should be aware of the following concepts in C++ 1) Pointers, references, arrays, temporaries, lvalue, rvalue 2)Polymorphism, static/dynamic binding, pure virtual functions & abstract class3) Operator overloading, function overloading, default arguments, friend functions 4) C++I/O, C++ File I/O, Stream manipulators 5) Able to convert a problem statement into pseudo code / algorithm and implement the same in C++.

-->**C++(L3)**-Should be working in a C++ Project. Should be aware of the

following concepts in C++ 1) Basics of Exception Handling 2) Namespaces, dynamic_cast, static_cast, const_cast, typeid & RTTI 3) C++ Containers, String Class, Basics of algorithms & iterators, Basics of Function objects

4) Template functions and classes, Basics of: Template Parameters, Restrictions on type parameters, template argument deduction, explicit and implicit instantiation, typename keyword 5) Class template specialization, default template arguments, Partial

Specialization 6) Template Compilation Models, Inclusion model, Separation Model 7) Function templates overloading, function template specialization 8) smart pointers / auto_ptr 9) Data Structures and Memory management in C++, different flavors of operator

new 10) Familiarity with development and debugging on a particular platform using appropriate tools 11) Familiarity with static

analysis tools like DeepCheck, C++ test etc.

-->**Application Testing(L3)**-Should be able to differentiate the different testing phases like Unit Integration, System, Acceptance, Regression testing and should be capable to perform these tests with the help of ready-test cases. Should understand Test Plans, Requirement Traceability Matrix, Orthogonal Array Tool. Should be able to do metrics analysis and reliability analysis effectively with DFA tool.

Education: Bachelors Degree Experience

Level : 5-8 YEAR

View:

[osc openFrameworks application-a92q-Qw22f4.mp4](#)

```
using namespace ci;
using namespace ci::app;
using namespace std;
```

```
class OSCSenderApp : public AppNative {
```

```
public:
    void setup();
    void update();
    void draw();
    void mouseMove( MouseEvent event );
    void mouseDrag( MouseEvent event );

    int                mMouseLocX;
    osc::Sender sender;
    float              positionX;
    std::string host;
    int                port;
};
```

```
void OSCSenderApp::setup()
```

```
{
    mMouseLocX = getWindowCenter().x;
    port = 3000;
    // assume the broadcast address is this machine's IP address but with 255 as the final value
    // so to multicast from IP 192.168.1.100, the host should be 192.168.1.255
    host = System::getIpAddress();
    if( host.rfind( '.' ) != string::npos )
        host.replace( host.rfind( '.' ) + 1, 3, "255" );
    sender.setup( host, port, true );
}
```

```
void OSCSenderApp::update()
```

```
{
    float freq = mMouseLocX / (float)getWidth() * 10.0f;
    positionX = cos(freq * getElapsedSeconds()) / 2.0f + .5f;

    osc::Message message;
    message.addFloatArg(positionX);
    message.setAddress("/cinder/osc/1");
    message.setRemoteEndpoint(host, port);
}
```

In the Google Drive:
OSCSenderApp.cpp

```
#include "cinder/app/AppNative.h"
```

```
using namespace ci;  
using namespace ci::app;
```

```
#include "OscListener.h"
```

```
class OSCListenerApp : public AppNative {  
public:
```

```
    void setup();  
    void update();  
    void draw();
```

```
    osc::Listener listener;  
    float positionX;
```

```
};
```

```
void OSCListenerApp::setup()
```

```
{  
    listener.setup( 3000 );  
    positionX = 0;  
}
```

```
void OSCListenerApp::update()
```

```
{  
    while( listener.hasWaitingMessages() ) {  
        osc::Message message;  
        listener.getNextMessage( &message );  
  
        console() << "New message received" << std::endl;  
        console() << "Address: " << message.getAddress() << std::endl;  
        console() << "Num Arg: " << message.getNumArgs() << std::endl;  
        for (int i = 0; i < message.getNumArgs(); i++) {  
            console() << "-- Argument " << i << std::endl;  
            console() << "---- type: " << message.getArgTypeName(i) << std::endl;  
            if( message.getArgType(i) == osc::TYPE_INT32 ) {  
                try {  
                    console() << "----- value: "<<  
message.getArgAsInt32(i) << std::endl;  
                }  
                catch ( ) {  
                    console() << "----- value: "<<  
                    message.getArgAsFloat(i) << std::endl;  
                }  
            }  
        }  
    }  
}
```

In the Google Drive:
OSCListenerApp.cpp

More topics in Cinder!

SECTION TWO: FLOCKING SIMULATION

Chapter 1: Camera and Parameters

We will learn about the Cinder Camera class and use it to create a 3D environment for the Particle engine we made in Section One. Then we will show how to setup a Params class for controlling variables during runtime.

Chapter 2: Rule One - Separation

The first rule of flocking is described and implemented. This rule states that all flocking objects should avoid getting too close to each other. This helps to mitigate overcrowding and collisions.

Chapter 3: Rule Two - Cohesion

The second rule is one of attraction. Flocking objects will move towards each other in order

CAMERA

In the previous tour, we only concerned ourselves with a 2D view of the *Particles*. They existed in their flatland, oblivious to the third dimension. It is time to throw an extra dimension at our *Particles* to see how they will behave in a 3D space. The best place to start in our new 3D project is the Cinder *Camera* class.

First, the include:

```
#include "cinder/Camera.h"
```

Then in the main class, we create a **CameraPersp**.

```
CameraPersp mCam;
```

The perspective *Camera* (as opposed to the orthographic *Camera*) is the standard way to move around in a virtual space. To describe our perspective *Camera lens*, we will use the **setPerspective()** method. Also note we prefix our member variables with the letter 'm' versus local variables which have no prefix. This is simply to make the code more readable later on. When you see variables with the 'm' prefix, you instantly know you are dealing with a member variable.

```
mCam.setPerspective( 60.0f, getWindowAspectRatio(), 5.0f, 3000.0f );
```

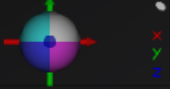
The camera's **setPerspective()** method takes four parameters. First is the horizontal **Field Of View**. The smaller the number, the tighter the viewing **frustum**. *(I usually choose a number between 60.0 and 90.0. There is some debate about the FOV of the human eye. It is estimated to be around 60 but the eye is very accommodating so this number varies. Also, if you take into account peripheral vision and the fact we have two eyes, the estimate would be as high as 140 to 180.)*

The second parameter is the aspect ratio of the application window. Cinder provides the **getWindowAspectRatio()** convenience method for getting the aspect ratio. If you prefer, you can calculate this yourself by taking the window width and dividing it by the window height.

The third and fourth parameters are for the clipping planes. The easiest way to think of it is this: Don't draw anything that is closer than the *near* clipping plane and don't draw anything that is further away than the *far* clipping plane. Since we are just looking at a bunch of particles that exist in a relatively confined area, we will choose values for the clipping plane that will accommodate our particles without also paying attention to a bunch of extra space where particles will likely never go. No reason to look all the way to infinity if nothing ever wanders more than 500 units from the camera.

Flocking

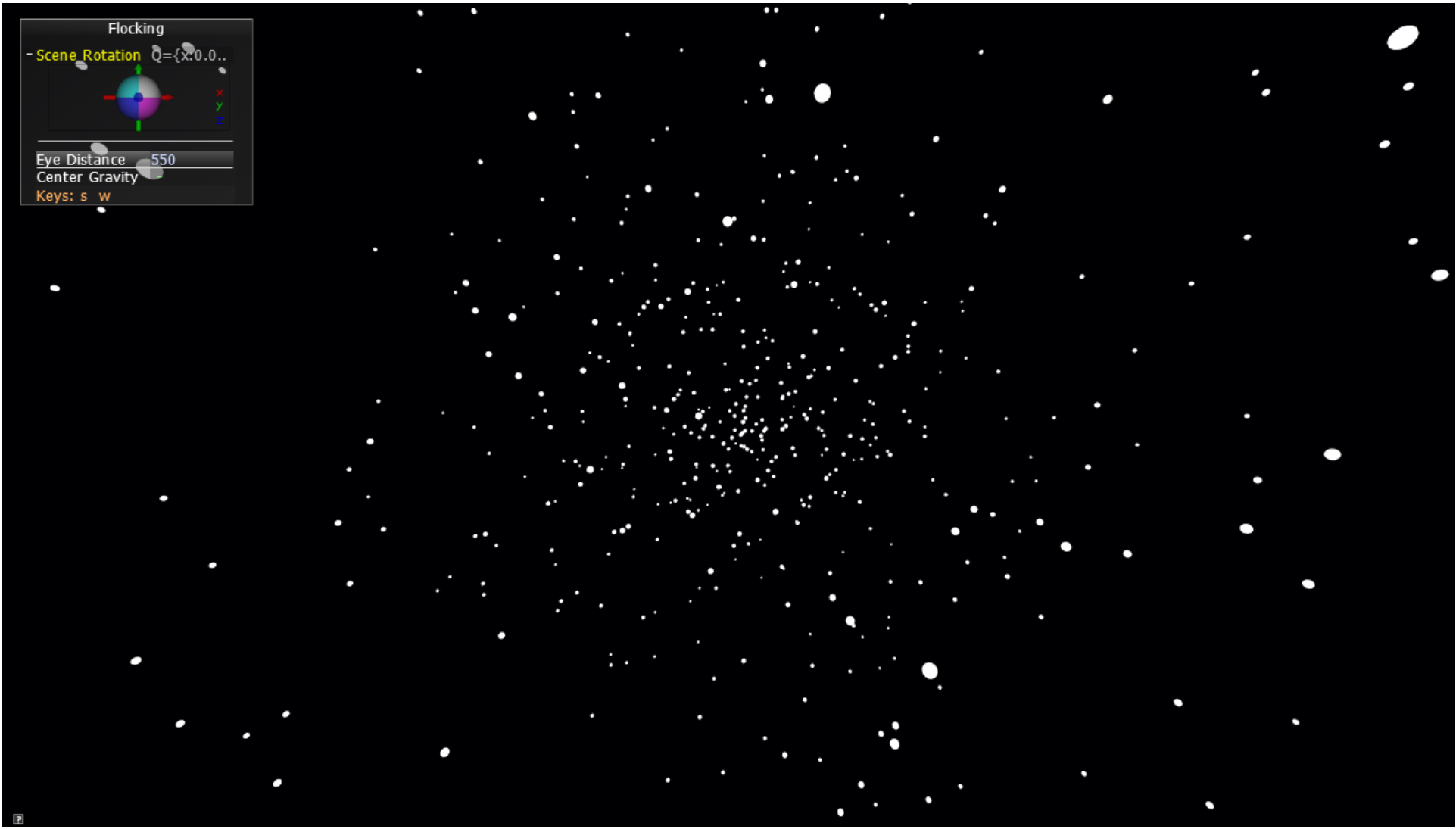
- Scene Rotation $Q=\{x:0.0..$



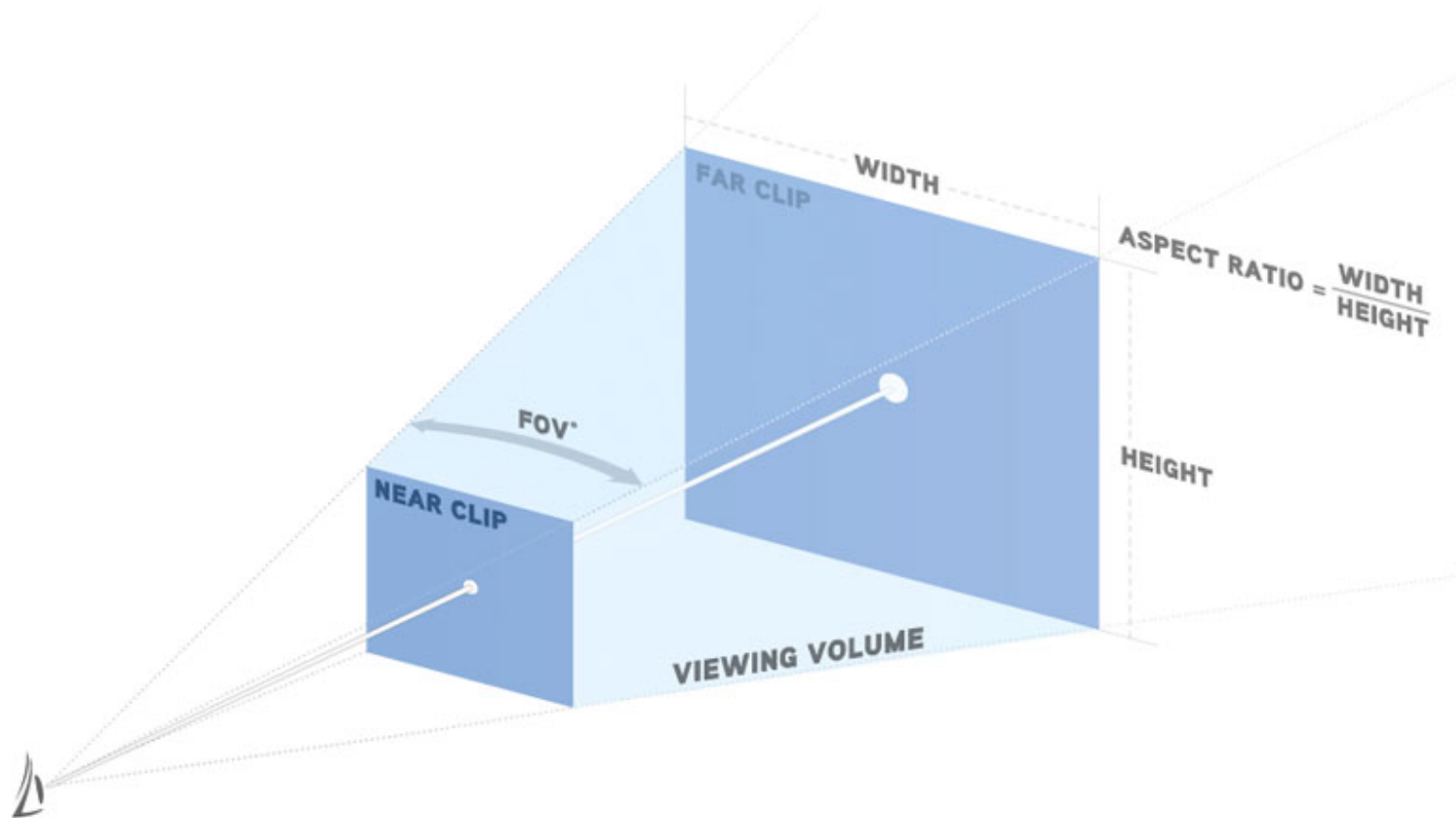
Eye Distance 550

Center Gravity

Keys: s w



Here is our camera setup explaining the `setPerspective()` parameters. Everything in the viewing volume will be shown in the application window.



Now that we have defined our camera, all that is left is for us to tell it where to be and where to look. This is definitely easier than it sounds but it will take a few lines to get it all squared away. First we need to create a `Vec3f`s for each of the three vectors the `CameraPersp` is expecting.

```
mEye = Vec3f( 0.0f, 0.0f, 500.0f );  
mCenter = Vec3f::zero();  
mUp = Vec3f::yAxis();
```

PARAMS

Hopefully by now you have played around with the Params class built around the [AntTweakBar](#) by [Antisphere](#). Params allows for an easy way to adjust variables in runtime with a minimal amount of setup. It really is surprisingly easy.

```
#include "cinder/params/Params.h"
```

After the include, make a new `cinder::params::InterfaceGl` `InterfaceGl` called `mParams`. It has three initial parameters: title, size and color (optional).

```
params::InterfaceGl mParams;  
mParams = params::InterfaceGl( "Flocking", Vec2i( 225, 200 ) );
```

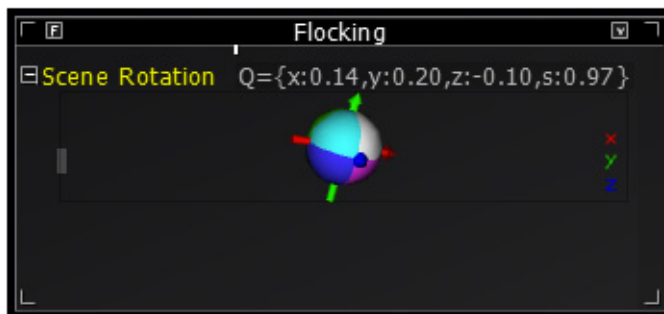
All we need now is to give it a reference of the variable we wish to control at runtime. In our case, we want a whole new variable that we can use to rotate the scene. We are going to invoke the power of the mighty [Quaternion](#). Don't be scared, you don't need to know the math behind the quaternion to enjoy some of the benefits.

```
Quatf mSceneRotation;
```

Then in `setup()`, after we initialize `mParams`, we add the following line.

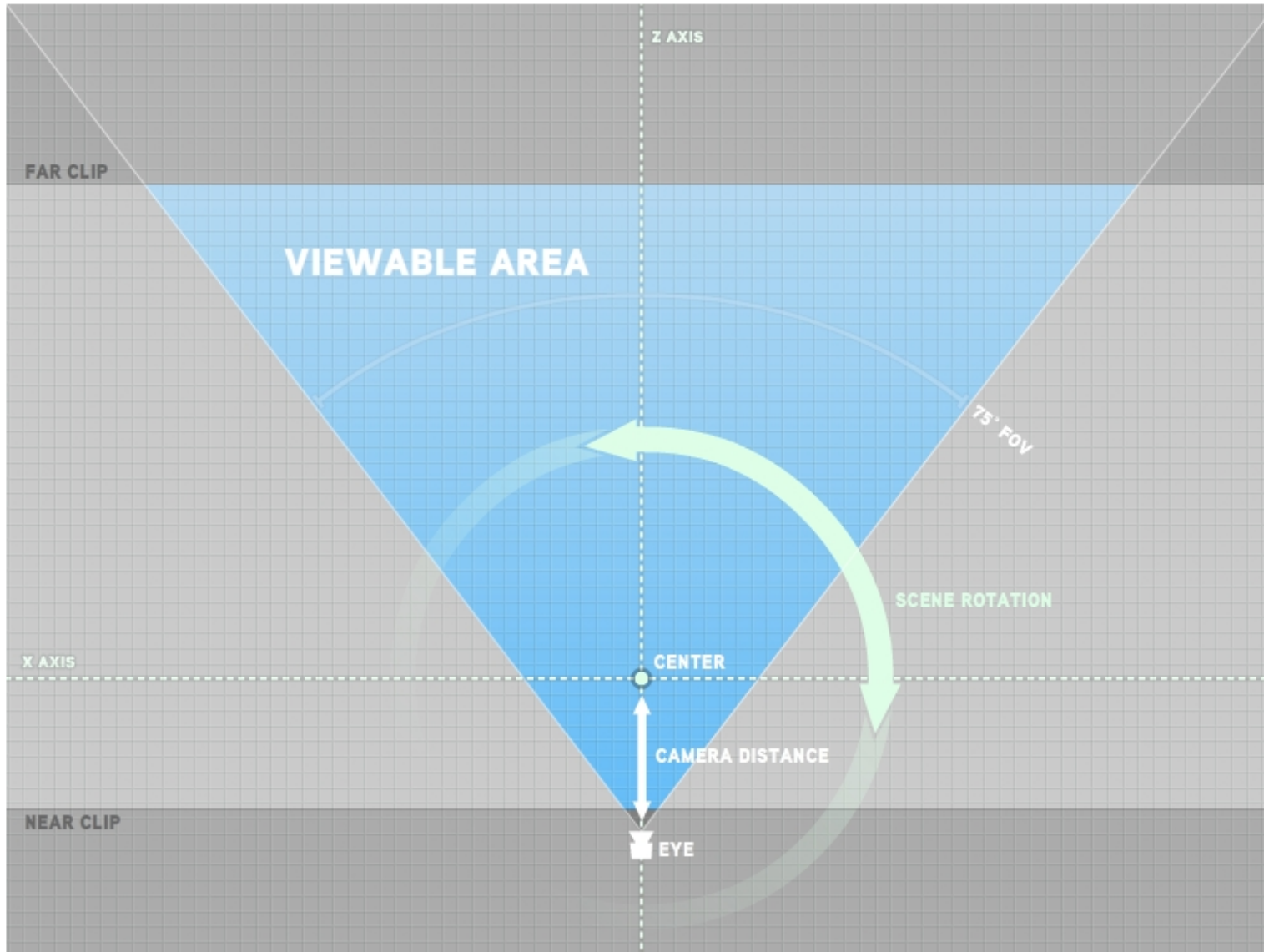
```
mParams.addParam( "Scene Rotation", &mSceneRotation );
```

With this line, now have the ability to control `mSceneRotation` in runtime using the `mParams` window. When adding a new tweakable parameter to your `InterfaceGl` instance, the `addParam()` method is expecting the memory address of your variable so it know where to look for its value. That's what the C++ Address-of operator (`&`) does. It grabs the memory address of a variable. Since we are asking `mParams` to give us control of a quaternion, it automatically does the right thing and gives us an arc-ball.



```
mCam.lookAt( mEye, mCenter, mUp );  
gl::setMatrices( mCam );  
gl::rotate( mSceneRotation );
```

When you click and drag in the params arc-ball miniwindow, the entire scene will rotate. Cinder does have an **Arcball** class which is explained in the ArcballDemo project in the samples folder. If you want more control over the arc-ball, that would be the place to start.



```

#include "cinder/app/AppBasic.h"
#include "cinder/Vector.h"
#include "cinder/Utilities.h"
#include "cinder/params/Params.h"
#include "cinder/Camera.h"
#include "ParticleController.h"

#define NUM_INITIAL_PARTICLES 500

using namespace ci;
using namespace ci::app;

class FlockingApp : public AppBasic {
public:
    void prepareSettings( Settings *settings );
    void setup();
    void update();
    void draw();

    // PARAMS
    params::InterfaceGlRef  mParams;

    // CAMERA
    CameraPersp             mCam;
    Quatf                   mSceneRotation;
    float                   mCameraDistance;
    Vec3f                   mEye, mCenter, mUp;

    ParticleController      mParticleController;

    bool                    mCentralGravity;
};

void FlockingApp::prepareSettings( Settings *settings )
{
    settings->setWindowSize( 1280, 720 );
    settings->setFrameRate( 60.0f );
}

```

```

void FlockingApp::setup()
{
    mCentralGravity = false;

    // SETUP CAMERA
    mCameraDistance = 500.0f;
    mEye             = Vec3f( 0.0f, 0.0f, mCameraDistance );
    mCenter          = Vec3f::zero();
    mUp              = Vec3f::yAxis();
    mCam.setPerspective( 75.0f, getWindowAspectRatio(), 5.0f, 2000.0f );

    // SETUP PARAMS
    mParams = params::InterfaceGl::create( "Flocking", Vec2i( 200, 160 ) );
    mParams->addParam( "Scene Rotation", &mSceneRotation, "opened=1" );
    mParams->addSeparator();
    mParams->addParam( "Eye Distance", &mCameraDistance, "min=50.0 max=1500.0 step=50.0 keyIncr=s keyDecr=w" );
    mParams->addParam( "Center Gravity", &mCentralGravity, "keyIncr=g" );

    // CREATE PARTICLE CONTROLLER
    mParticleController.addParticles( NUM_INITIAL_PARTICLES );
}

void FlockingApp::update()
{
    // UPDATE CAMERA
    mEye = Vec3f( 0.0f, 0.0f, mCameraDistance );
    mCam.lookAt( mEye, mCenter, mUp );
    gl::setMatrices( mCam );
    gl::rotate( mSceneRotation );

    // UPDATE PARTICLE CONTROLLER
    if( mCentralGravity ) mParticleController.pullToCenter( mCenter );
    mParticleController.update();
}

void FlockingApp::draw()
{
    gl::clear( Color( 0, 0, 0.01f ), true );
    gl::enableDepthRead();
    gl::enableDepthWrite();

    // DRAW PARTICLES
    glColor4f( ColorA( 1.0f, 1.0f, 1.0f, 1.0f ) );
    mParticleController.draw();

    // DRAW PARAMS WINDOW
    mParams->draw();
}

```



```

#include "cinder/app/AppBasic.h"
#include "cinder/Rand.h"
#include "cinder/Vector.h"
#include "ParticleController.h"

using namespace ci;
using std::list;

ParticleController::ParticleController()
{
}

void ParticleController::pullToCenter( const ci::Vec3f &center )
{
    for( list<Particle>::iterator p = mParticles.begin(); p != mParticles.end(); ++p ) {
        p->pullToCenter( center );
    }
}

void ParticleController::update()
{
    for( list<Particle>::iterator p = mParticles.begin(); p != mParticles.end(); ++p ) {
        p->update();
    }
}

void ParticleController::draw()
{
    for( list<Particle>::iterator p = mParticles.begin(); p != mParticles.end(); ++p ) {
        p->draw();
    }
}

void ParticleController::addParticles( int amt )
{
    for( int i=0; i<amt; i++ ) {
        Vec3f randVec = Rand::randVec3f();
        Vec3f pos = randVec * Rand::randFloat( 50.0f );
        Vec3f vel = randVec * Rand::randFloat( 5.0f );
        mParticles.push_back( Particle( pos, vel ) );
    }
}

```

```
#include "cinder/app/AppBasic.h"
```

```
using namespace ci;
```

```
Particle::Particle()
```

```
{  
}
```

```
Particle::Particle( Vec3f pos, Vec3f vel )
```

```
{  
    mPos = pos;  
    mVel = vel;  
    mAcc = Vec3f::zero();  
  
    mRadius = 2.0f;  
    mDecay = 0.99f;  
}
```

```
void Particle::pullToCenter( const Vec3f &center )
```

```
{  
    Vec3f dirToCenter = mPos - center;  
    float distToCenter = dirToCenter.length();  
    float maxDistance = 300.0f;  
  
    if( distToCenter > maxDistance ){  
        dirToCenter.normalize();  
        float pullStrength = 0.0001f;  
        mVel -= dirToCenter * ( distToCenter - maxDistance ) * pullStrength ;  
    }  
}
```

```
void Particle::update(){
```

```
    mVel += mAcc;  
    mPos += mVel;  
    mVel *= mDecay;  
    mAcc = Vec3f::zero();  
}
```

```
void Particle::draw()
```

```
{  
    gl::drawSphere( mPos, mRadius, 16 );  
}
```

What is open frameworks and why should you care? (Btw wikipedia is useful)

History [\[edit\]](#)

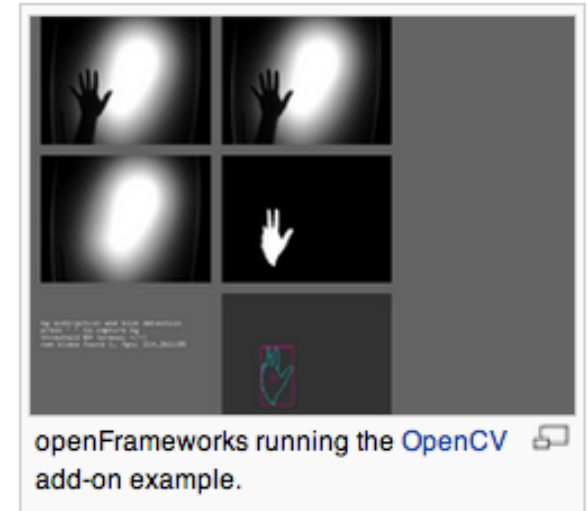
OpenFrameworks v0.01 was released by Zachary Lieberman on August 3, 2005. By February 2006, version v0.03 was in use by Lieberman's students at the [Parsons School of Design](#), New York City. According to its authors, openFrameworks was developed:

(for) folks using computers for creative, artistic expression, and who would like low level access to the data inside of media in order to manipulate, analyze or explore. That audience we felt was significantly underserved by the current crop of C++ libraries. ^[1]

Related projects [\[edit\]](#)

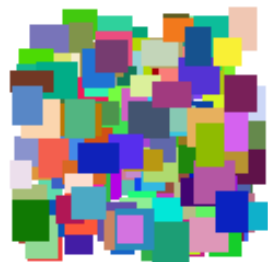
Its emphasis on "creative" uses draws parallels to [Processing](#) as both projects present a simplified interface to powerful libraries for media, hardware and communication. openFrameworks's main difference from [Processing](#) is that it is written in [C++](#), instead of [Java](#). Users will find many similarities between the two libraries, for example what is `beginShape()` in Processing is `ofBeginShape()` in openFrameworks. The openFrameworks wiki includes an article for people coming to openFrameworks from Processing.^[2]

Another similar project is [Cinder](#), which is also a C++ library framework for creative programming. The primary difference is that openFrameworks has a larger number of dependencies on open source libraries, allowing advanced programmers more control and transparency, while Cinder is more dependent on libraries built into the operating systems it sits on top of, which generally means updates and bug fixes are more frequent and reliable.

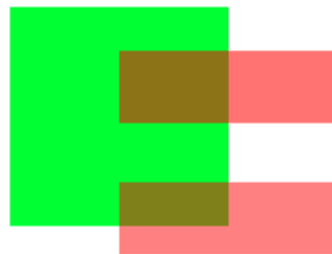




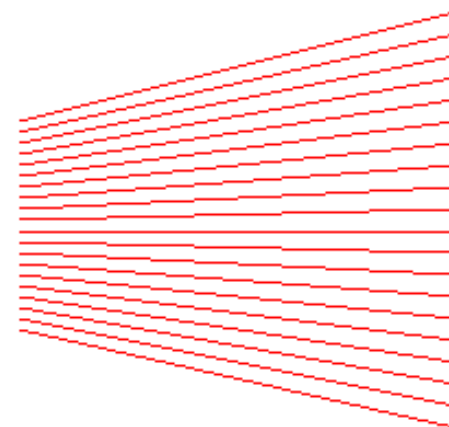
circle



rectangles



transparency



lines
press 's' to toggle smoothness

```

//-----
void ofApp::draw(){

    //----- circles
    //let's draw a circle:
    ofSetColor(255,130,0);
    float radius = 50 + 10 * sin(counter);
    ofFill(); // draw "filled shapes"
    ofCircle(100,400,radius);

    // now just an outline
    ofNoFill();
    ofSetHexColor(0xCCCCCC);
    ofCircle(100,400,80);

    // use the bitMap type
    // note, this can be slow on some graphics cards
    // because it is using glDrawPixels which varies in
    // speed from system to system. try using ofTrueTypeFont
    // if this bitMap type slows you down.
    ofSetHexColor(0x000000);
    ofDrawBitmapString("circle", 75,500);

    //----- rectangles
    ofFill();
    for (int i = 0; i < 200; i++){
        ofSetColor((int)ofRandom(0,255),(int)ofRandom(0,255),(int)ofRandom(0,255));
        ofRect(ofRandom(250,350),ofRandom(350,450),ofRandom(10,20),ofRandom(10,20));
    }
    ofSetHexColor(0x000000);
    ofDrawBitmapString("rectangles", 275,500);

    //----- transparency
    ofSetHexColor(0x00FF33);
    ofRect(400,350,100,100);
    // alpha is usually turned off - for speed puposes. let's turn it on!
    ofEnableAlphaBlending();
    ofSetColor(255,0,0,127); // red, 50% transparent
    ofRect(450,430,100,33);
    ofSetColor(255,0,0,(int)(counter * 10.0f) % 255); // red, variable transparent

```

In the Google Drive:
OpenFrameworks Graphics src

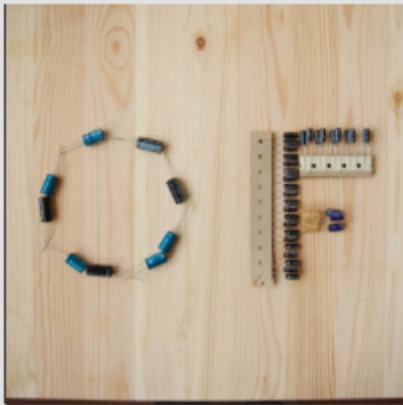
Lots of “add ons” included in OpenFrameworks including

- Video
- PDF
- OSC
- OpenCV (Computer Vision)
- SVG reader
- Etc.

PDF OUTPUT EXAMPLE

press r to start pdf multipage rendering
press s to save a single screenshot as pdf to disk

Images can also be embedded into pdf



e

drop images here

drop images here

TTF Font embedded into pdf as vector shapes

Current Frame: 1981



Save Screen as PDF

```
//-----  
void ofApp::draw(){  
    if( oneShot ){  
        ofBeginSaveScreenAsPDF("screenshot-"+ofGetTimestampString()+".pdf", false);  
    }  
  
    ofSetColor(54);  
    ofDrawBitmapString("PDF OUTPUT EXAMPLE", 32, 32);  
    if( pdfRendering ){  
        ofDrawBitmapString("press r to stop pdf multipage rendering", 32, 92);  
    }else{  
        ofDrawBitmapString("press r to start pdf multipage rendering\npress s to save a single screenshot as pdf  
to disk", 32, 92);  
    }  
  
    ofFill();  
    ofSetColor(54,54,54);  
    ofDrawBitmapString("TTF Font embedded into pdf as vector shapes", 32, 460);  
  
    if( oneShot || pdfRendering ){  
        font.drawStringAsShapes("Current Frame: ", 32, 500);  
        ofSetColor(245, 58, 135);  
        font.drawStringAsShapes( ofToString(ofGetFrameNum()), 32 + font.getStringBoundingBox("Current  
Frame: ", 0, 0).width + 9, 500);  
    }else{  
        font.drawString("Current Frame: ", 32, 500);  
        ofSetColor(245, 58, 135);  
        font.drawString( ofToString(ofGetFrameNum()), 32 + font.getStringBoundingBox("Current Frame: ", 0, 0).width + 9, 500);  
    }  
}
```


The first project to OpenCV

1. Creating a Project

We assume that Microsoft Visual C++ 2008 Express Edition and OpenCV 2.1 is already installed.

1. Run VS2008

2. Create a console project

File - New - Project - Win32 Console Application, in the Name enter Project1, click OK.

3. Set up the path

Alt + F7 - opens the project properties

Configuration Properties - C / C++ - General - Additional Include Directories, where we put the value "C:\Program Files\OpenCV2.1\include\opencv";

Linker - General - Additional Library Directories, where we put the value of C:\Program Files\OpenCV2.1\lib\

Linker - Input - Additional Dependencies -

cv210.lib cvaux210.lib cxcore210.lib cxts210.lib highgui210.lib for Release,
cv210d.lib cvaux210d.lib cxcore210d.lib cxts210.lib highgui210d.lib for Debug

The first project to OpenCV

2. Reading the image and display it on screen

1. Preparing the input data:

file http://www.fitseniors.org/wp-content/uploads/2008/04/green_apple.jpg

write in C:\green_apple.jpg

2. Writing in Project1.cpp:

```
# Include "stdafx.h"
```

```
# Include "cv.h"
```

```
# Include "highgui.h"
```

```
using namespace cv;
```

```
int main (int argc, const char ** argv)
```

```
{
```

```
Mat image = imread ("C:\\green_apple.jpg");// Load image from disk
```

```
imshow ("image", image); // Show image
```

```
waitKey (0); // Wait for keystroke
```

```
return 0;
```

```
}
```



3. Press F7 - compilation, F5 - run.

The program will show the image in the window and by pressing any key will complete its work.

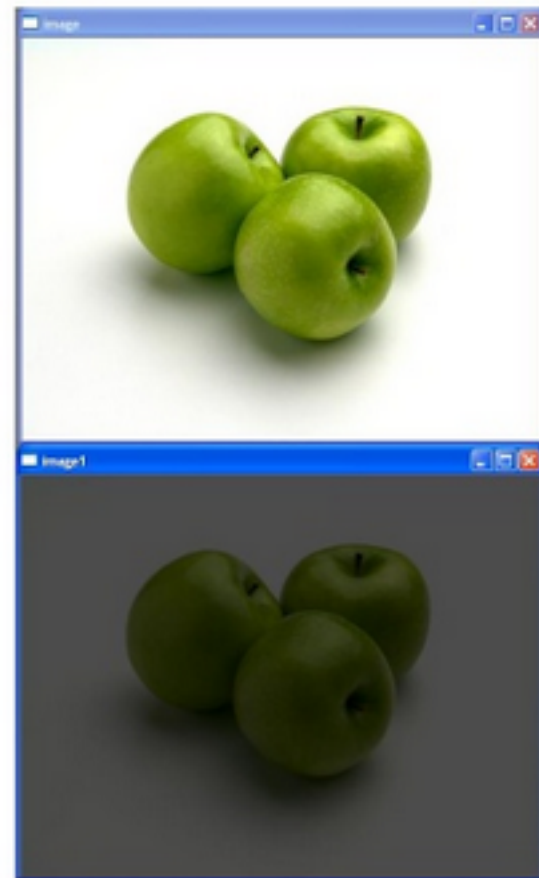
The first project to OpenCV

3. Linear operations on images

Replace the text in the main from the previous for example:

```
int main (int argc, const char ** argv)
{
    Mat image = imread ("C:\\green_apple.jpg");

    // Image1 pixel by pixel is equal to 0.3 * image
    Mat image1 = 0.3 * image;
    imshow ("image", image);
    imshow ("image1", image1);
    waitKey (0);
    return 0;
}
```



And more add-ons

```
#include "ofMain.h"
#include "ofxCv.h"
#include "ofxNetwork.h"
#include "ofxOsc.h"
// #include "ofxSynth.h"
#include "ofxXmlSettings.h"
#include "ofx3DModelLoader.h"
#include "ofxAssimpModelLoader.h"
#include "ofxThreadedImageLoader.h"

class testApp : public ofApp{
public:
    void setup();
    void update();
    void draw();
    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    // we don't actually use these
    // just checking to see if they
    // all work in the same place :)

    ofxCvGrayscaleImage cvGray;
    ofxTCPClient client;
    ofxTCPServer server;
    ofxOscSender osc_sender;
    ofxXmlSettings settings;
    ofx3DModelLoader modelLoader;
    ofxAssimpModelLoader betterModelLoader;
    //ofxSynth synth;
    ofxThreadedImageLoader threadedLoader;
};
```

Final project presentations June 10 at Turbine!

How are things coming together?

Syllabus

*** SNIP ***

10) Gesture recognition & depth controllers like the Microsoft Kinect, Network Programming & TCP/IP, OSC May 27

11) Selected Topics June 3

12) Working on student projects - June 10

Final project presentations Project 3/Final Project Due June 10