KronicSpace Final Report

Justin Kent

Jon Pearce

Toby Segaran

Abstract

KronicSpace is a 3-dimensional, customizable, extendable, modular universe. It provides a framework for creating self-perpetuating animations, in which objects with special behaviors can interact with each other, their environment, and the user. We implemented KronicSpace using Java3D. Using the KronicSpace framework, we created two self-perpetuating, interactive animations: an aquarium and a battle.

Introduction

We decided that current animation tools are too limiting, because they do not provide for a self-perpetuating environment. Once an animation is created, it is static, and its member objects do not interact with one another but simply act under their author?s direction. Therefore, we made it our goal to produce a system containing objects with user-defined behaviors to interact with their environment and each other. To make this even more interesting, the environment itself would have behaviors and would also be able to respond to outside stimuli such as the computer keyboard or audio input. KronicSpace will make it easy to produce a continuous animation in which the objects are truly interacting with one another, not just appearing to interact, as in traditional animation.

Goals

To focus our efforts, we decided upon the following goals for KronicSpace:

Generality

The KronicSpace framework should be general enough to create many different kinds of 3D environments. The kinds of actions that objects can do should be enough to cover almost any kind of scene someone might want to create, and should not be biased toward any particular genre.

Interactivity

KronicSpace should provide a workable way to create interactive scenes. In these scenes, the environment would receive outside input, such as sound or user keystrokes, and pass relevant information to objects that might react to this input.

Simplicity

The most important goal we had for KronicSpace was simplicity. Too many arcane, complex behavior descriptions would make it difficult to create environments and would hamper KronicSpace development. Designing the system with simplicity in mind meant sticking to a minimum number of object actions, and filtering all user input through the single environment object. With KronicSpace, anyone with a little Java knowledge should be able to create interesting, self-perpetuating 3D environments.

Demonstratability

Finally, as well as creating the KronicSpace framework, we wanted to create some actual environments that would show off KronicSpace?s capabilities. Our original goal, which we did implement, was to create an aquarium with different kinds of fish interacting with the user and each other.

Achievements

Meeting the goals

KronicSpace met the goals of generality, interactivity, simplicity, and demonstratability. Our system of objects and actions makes it quite easy to create many different kinds of objects and define their behaviors. For example, here is the code for the behavior of one of the objects in the aquarium animation: the fish egg.

```
public Action clockTick(ObjectInformation me,Vector listOfObjects) {
    maturity++;
    if (maturity>=100 && maturity<110) {
        if ((maturity & 1)!=0) {
            return new Action("MOVE","",-0.001f,0f,0f);
        } else {
            return new Action("MOVE","",0.001f,0f,0f);
        }
    }
    if (maturity==116) {
        return new Action("MOVE","",0f,0f,0f);
    }
    if (maturity==117) {
        return new Action("SPAWN","Goldfish",me.getx(),
                me.gety(),me.getz(),
```

```
                    0f,0f,10f,0.1f);
        }
        if (maturity==122) {
            return new Action("DIE","",0,0,0);
        }
        return new Action("NOTHING","",0f,0f,0f);
    }
```

According to this code, fish eggs will do nothing until their maturity counter reaches 100. Then they will move around for 10 clock ticks, and then spawn a fish and die.

Once the objects have been created, setting up a KronicSpace scene is also easily done using a .space file. Here is an excerpt from the aquarium environment *aquarium.space*:

```
[OBJECT]
class=Goldfish
x=0
y=-0.2
z=0.2
rotx=0
roty=0
rotz=0
scale=0.2
[/OBJECT]
```

[OBJECT]

class=FishFeeder

x=0

y=0.7

z=0

scale=0.8

[/OBJECT]

[OBJECT]

class=crazyshark

x=0.2

y=0

z=0

rotx=0

roty=0

rotz=0

scale=0.003

[/OBJECT]

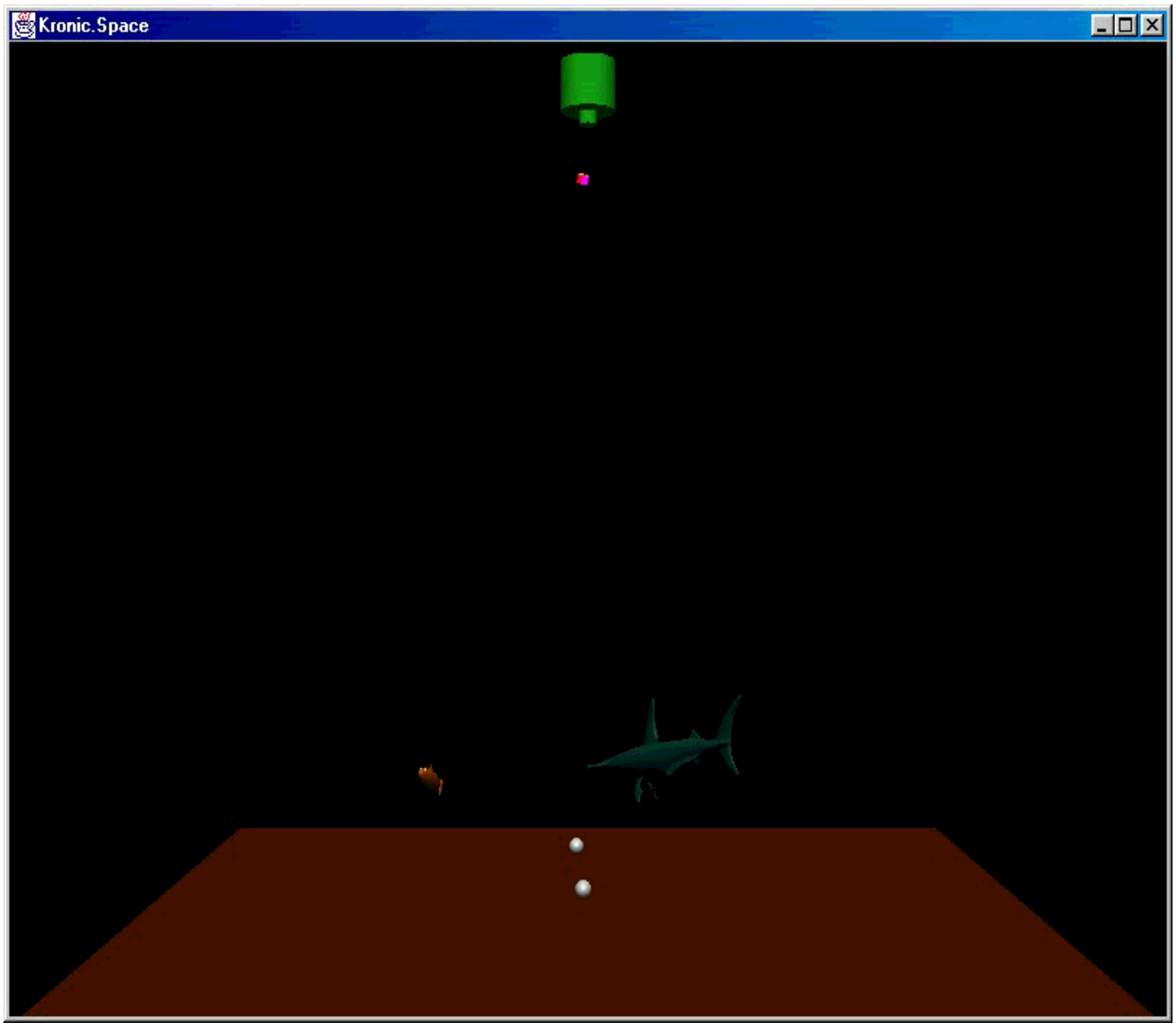
To set up a scene, each object is declared, and the appropriate attributes (position, rotation, scale, etc.) are adjusted or defined if necessary.

KronicSpaces

Using the framework we developed, we created two self-perpetuating KronicSpace environments: an aquarium and an aircraft battle.

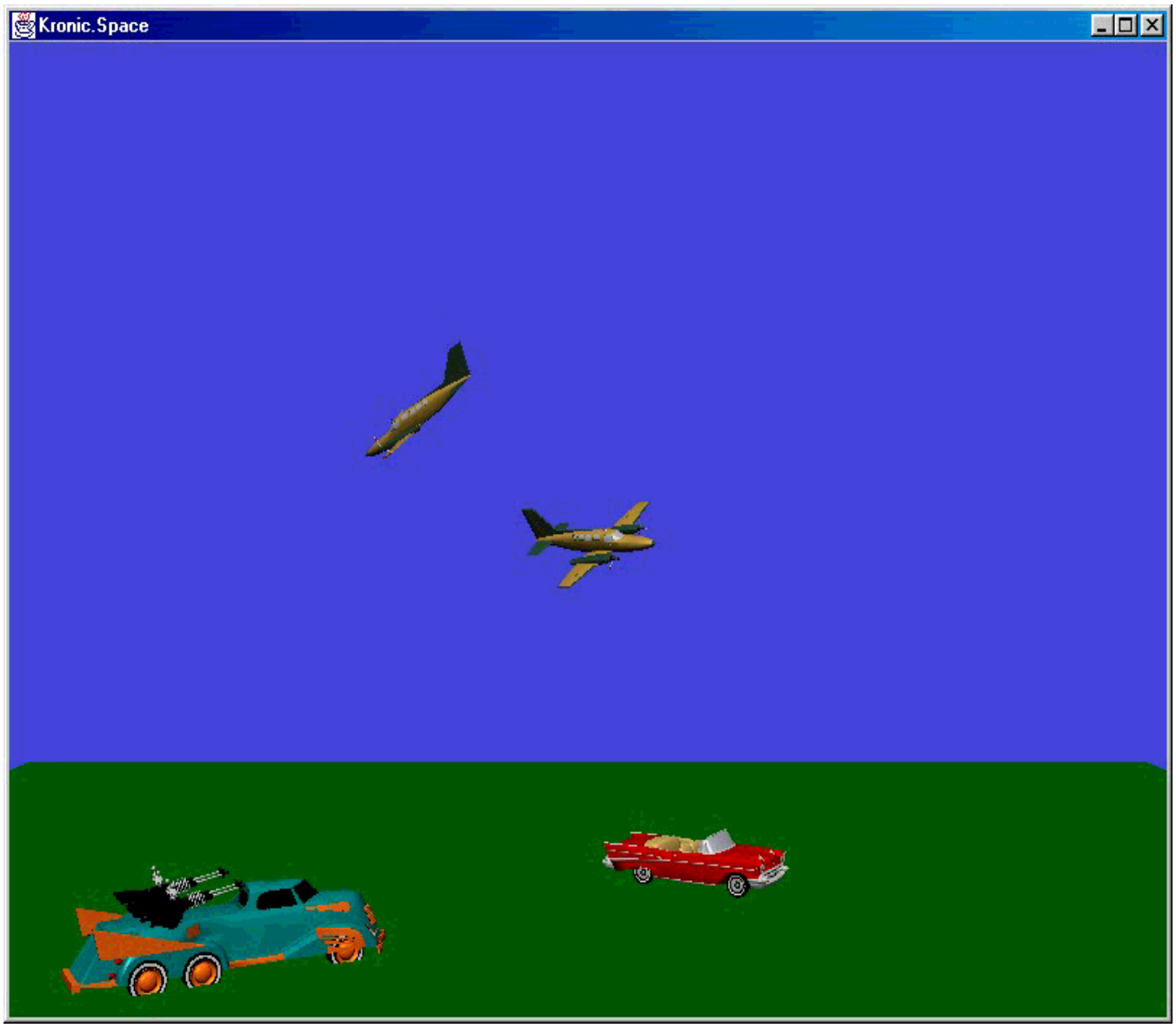
Aquarium

This environment contains some fish that start out swimming around randomly. Every so often, they get the urge to lay eggs, which drop to the bottom of the aquarium. After a while, these eggs shake and then hatch tiny fish, who gradually grow into full-sized fish. Meanwhile, the shark chases after the fish, always following the nearest one, until it catches up to it and eats it. The user can feed the fish at any time, pressing the spacebar to drop food out of the fish-feeder object at the top of the aquarium. When the food is dropped, the fish will swim up to the food and eat it.



Battle

This environment is more like a game. Cessna planes fly around randomly in the sky. The user can control a Chevy Bel Aire that can drive around and shoot heat-seeking missiles at the planes. When a missile hits a plane, it explodes.



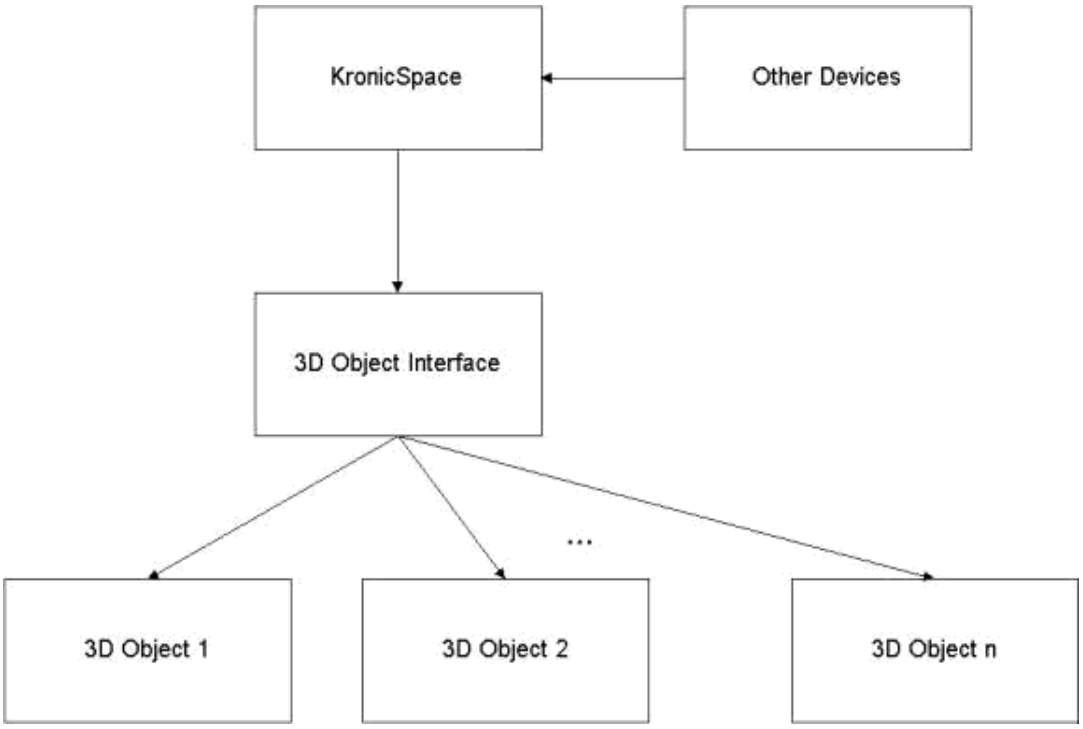
Program structure

KronicSpace is written in Java 1.2. Essentially, the main program is given a file containing information about the space and a list of 3D objects that inhabit it. It then loads the classes for the corresponding objects and places them within the space. The objects, which control their own behavior, would then

interact and produce a continuous animation.

The objects are implemented according to an interface. This interface allows KronicSpace to query the objects for information about what they look like, which direction they would like to move and how fast, their rotation angle and scaling, and whatever messages they want to convey to the environment. It also allows KronicSpace to pass information to the object such as proximity and collisions with other objects, clock ticks, current location, and environmental factors.

This diagram shows the fundamental components of KronicSpace and their relationships.



KronicObject

This is the interface through which objects interact with each other and the environment. It follows this structure:

BranchGroup MyAppearance()

This procedure is called by KronicSpace whenever it needs to know the 3D model of this object. The

appearance can change according to different factors, or it can remain constant through the life of the object. The procedure returns a scene graph specifying the appearance of this object, centered at the origin and being bounded by a certain area.

Action ClockTick (ObjectInformation me, Vector listOfObjects)

This procedure is called at a fixed interval of once every 100ms. It passes information about this object in "me". The return value specifies the action that the object would like to do at this time. The procedure is also passed the list of other objects in the scene so it can decide how to react with them.

Action Collision (ObjectInformation me, ObjectInformation other)

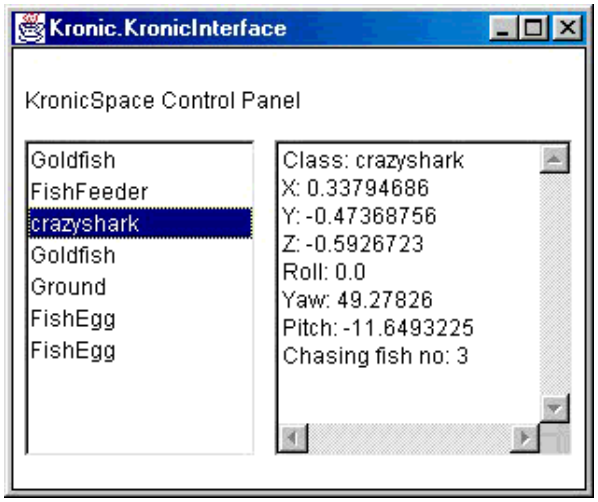
Called when an object collides with another.

Action EnvironmentMessage (Integer Intensity, String Information)

Called when the selected input device acts to stimulate the environment. The intensity and information parameters are information regarding specific devices. For example, the intensity could be the strength of a music beat, or information could specify which key was pressed.

KronicInterface

This class is the user interface that goes along with KronicSpace. It lists all the objects in a scene, and the user can click on each item in the list to find current information about an object.



Space

This is the main program that is run from the command line. The syntax for using this is:

Java Kronic.Space *spacefile*

ObjectInformation

This class keeps track of the various attributes of an object. It includes the following information:

- current XYZ coordinates
- current velocity vector
- current rotation
- current rotation velocity vector
- scaling factor
- a message string
- a Boolean value indicating whether or not it is scheduled to die during this clock tick.

All this information can be accessed with helper functions, and can be updated given a clock tick.

Action

This class represents an action that an object can take at a clock tick. The possible actions are:

· NOTHING - do nothing.

· MOVE - move according to a specified XYZ vector.

· FORWARD - move forward given a current speed.

· SPIN - rotate according to a specified XYZ vector.

· SCALE - change size according to a scale factor.

· SPAWN - spawn another object of the same type or of a different type.

· DIE - cease to exist in the KronicSpace.

The action class contains the string that tells what the action is, as well as appropriate floating point parameters.

KronicFunction

This is a special class that we created, with several useful mathematical functions that objects can use to perform common tasks like chasing other objects and detecting proximity. Included among these are:

GetDesiredZRotation

Given an object and a point in space, this function calculates and returns the yaw that an object should have in order to face towards the point.

GetDesiredYRotation

Given an object and a point in space, this function calculates and returns the pitch that an object should have in order to face towards the point.

nearest

Given an object and the class name of a different type of object, this function returns the closest object of the given class. This is useful, for example, for the shark wishing to chase the closest fish.

distanceBetweenObjects

Given two objects, this function returns the distance between them

withinSphere

Given an object and a center-point and radius of a sphere, returns true if the object is within the sphere and false otherwise.

withinBox

Given an object and a center-point and edge-length of a box, returns true if the object is within the box and false otherwise.

Challenges

Of course, the key challenge with this project was learning to use all the tools and come up with an engineering design that would make our rather ambitious idea possible. In addition, there were many more specific problems that made this project a challenge.

Actually applying the vector math that we were taught in class to specific problems, like making the objects appear in the right place and move around according to their rotations as well as follow other objects, proved to be more challenging than we had initially anticipated. Overcoming these problems involved several hours of pencil-and-paper calculation, and great care in implementation that the correct order of matrix operations was preserved.

Creating models using the Java 3D language also proved to be very painstaking, so we had to find a better way to produce models, since we really wanted to focus on the design and interaction aspects of this project. To solve this, we managed to find a library, which would load Alias Wavefront files. Thus we were able to use Wavefront to make models and also download models from the web to use in our animations.

Another big challenge was getting the music to work properly. The primary problem is that the sound library will only work with JDK1.3, which is not yet available on Athena. We did manage to get it working on our Windows machines although not as comprehensively as we would have liked.

Individual Contributions

The design of KronicSpace was done collaboratively. We brainstormed together and came up with the general idea, and then decided on the different actions that objects would be able to do, and how the objects and environment would interact.

Justin

Justin came up with the battle animation. He also defined the behavior for most of the objects in that animation, as well as some of the objects in the aquarium animation. He also worked with JavaSound to introduce music to KronicSpace.

Jon

Jon did the mathematics work for the math helper class. He also found or modeled all of the models for the various animations, and worked on the ObjectInformation and Action classes with Toby.

Toby

Toby implemented the underlying framework for KronicSpace. He also defined the behavior for most of the objects in the aquarium animation.

Lessons Learned

In the design and development of KronicSpace, we learned a good deal about the practical side of graphics programming.

Java3D

Before this project, none of us had used a 3D graphics programming library except the Inventor we learned in class. By the end, we were all well-versed in Java3D, a 3D graphics library for Java 1.2.

Physics/Math

In class we learned the correct matrix operations necessary for object scaling and transformations, as well as rotation (roll, pitch, and yaw). In this project, we had the chance to shuffle through our lecture notes, find that material, and apply it. In addition, we also had to figure out the vector mathematics to make one object chase another, which we used in the aquarium environment.

Project Management

This project also improved our project management skills. Before now, only one of us had worked on a large, multi-person software project. We learned the importance of designing the program specifications early and well. Since we all knew what the specifications were, for the most part, it was easier to break up the work and have it all fit together in the end. For example, Jon could do the math helper class while Toby was working on the KronicSpace framework and Justin was creating objects for the aquarium animation. After this project, we feel that we can confidently plan and execute a larger programming endeavor.

Acknowledgements

Some models found at:

www.rccad.com/Gallery.htm

www.3dcafe.com/asp/animals.asp

Bibliography

Bouvier, Dennis J. *Getting Started with the Java3D API.* Sun Microsystems, 1999.

Java3D API Specification.

java.sun.com/products/java-media/3D/forDevelopers/j3dapi/index.html