

xTNZ

- A three-dimensional pc based ecosystem -

Rui Filipe Antunes
Goldsmiths College
University of London

October 31, 2006

Abstract

*xTNZ*¹, is focused on the exploration of the possibilities of using artificial life in the context of art. My aim is to develop an ecosystem based on a real-time three-dimensional PC based system sustaining a “*living*” virtual environment. The entities populating this virtual world have been designed to be active and responsive. They behave and interact with each other, they reproduce according to eventual interactions and they change their own properties (such as visual appearance or dimensions). An unpredictable visual representation of the world will emerge, shapes will evolve in time according to the creatures interaction.

1 Introduction

In contemporary digital culture, the social status of the computer has risen as a response to the development of its performance. From video games, media interface and day-to-day working tools, the computer has gained in the last decade a growing influence in a social context where visual information is privileged rather than other senses.

In the visual arts domain, new powerful tools are emerging through the creation of visual effects, live performances, human detection in interactive works, and a wide range of artistic approaches². The integration of these new techniques³, on the one hand, offers a renewed pictorial richness to explore. On the other hand, by using its own language, integration widens the possibility of developing artworks which are sociologically relevant in the present cultural context of a spreading dependency on computers and videogames⁴. In this context, artificial life or real time three-dimensional images are symptomatic examples of these new techniques available nowadays simply in personal computers.

By placing a personal computer in an exhibition space, this object is by itself charged with different connotations and significations. As objects, personal computers - as in the case of television in the late 60's-, have become an ordinary element in our houses. Users/viewers have integrated the design of computers in their lives. They know what to expect from them and how to manipulate them to obtain certain results.

However, despite all the cultural and productive richness introduced by the Information Technologies revolution, this increasing influence (mostly caused by Internet and video games) is prompting a new sociological phenomenon: computer alienation. Humanity is creating new worlds, extensions from the empiric reality⁵. The problem would be that while we experience these virtual dimensions, we are losing touch with empiric reality. These virtual experiences prevent contact between the body, in its whole sensitive experience and its environment. Baudrillard, a “theorist of the computer screen”, describes an audience that is absent, absorbed into the PC-monitor, losing their own image and predicting the disappearance of reality. Within the “extended body” we have access to a rich network of all kind of databases and sensations promoted by mental stimuli. However, a price is paid by neglecting the whole experience of the organic body. Human civilization is heading toward Moravec’s nightmarish vision of a future of brains imbedded in computers⁶, in contradiction with enactivist theories conceiving the whole body as a source of cognition [O’Regan/Noe].

As a principle, when a medium is explored, the nature of the tool is part of the final work. An art piece using three-dimensional virtual environments suggests an interrogation of the status of a 'virtual world' as a material entity. According to Grau ⁷, a virtual world is simply an algorithm producing a continuous flow of images (approximately between 15 to 30 frames per second) in order to create an illusionary cortical effect of an existing living thing. In the conceptualization of this work, I have retained two concepts from this idea: firstly, that a computer based database manipulated through an algorithm produces an imagery; and secondly, that it produces an illusionary space on the screen - with virtual reality, this illusion is more notorious for the fake three-dimensionality on a flat surface.

Let's take the idea of the computer as an object that produces and hosts a database to create images, World agents have both a numerical and internal representation in the database as well as a geometrical and external visual representation. Interacting with the virtual world, the users/viewers are interacting with the database itself. Indeed they are visiting a database in its formal representation of geometrical shapes. The virtual world is a living database; a continuous visual representation of data.

Following Manovich's argument of a content-interface understood as a conceptual unified whole ⁸, I have chosen to present this work in a personal computer. Since, xTNZ is a database residing in a computer and running on its memory, I believe that it should be presented in this manner to the public - directly through a computer not mediated by any other device.

Moreover, the fact of having an object in the showroom, the computer, offers an outside/inside duality between a viewer outside the object and a database inside it. This duality creates the perfect conditions to develop an art piece which works with the concepts introduced previously of an external reality accessed via a computer and the body detachment involved in this action.

The symbolic world I have chosen to represent the extended reality offered by computer technologies is a symbolic three-dimensional world with living creatures in a utopian ecosystem. These creatures have human sounds such as kissing or chewing; and are visually represented with images from human tissues - from internal organs, muscles and bones - digitally manipulated and applied as textures; ironically emphasizing the disembodiment I have been referring to.

On designing xTNX, I am using the common cliché of creating organic forms to compensate the machine's logic and mathematics. I am implementing the natural behaviors of natural life such as breeding, reproducing and dieing. In this way, I can play with the immediate metaphor of a world and the idea of an external disturbance (the user/viewer). I have been inspired by real life ecosystems, where external agents introduce disturbance on the natural habitats, bringing about disturbance of the natural balance. This can be extended from biological habitats to urban habitats where the contemporary reality of global politics, culture and tourism induces external sociological changes. Thereby, I have incorporated in this work the idea that original places are disturbed by external visits. The user (viewer) is considered an intruder to the system. His presence (as moving by touching the keyboard) disturbs the overall balance disabling the agent's reproductive cycle. If staying too long, the intruders can even provoke the system destruction.

In summary, with xTNZ my aims are:

- To investigate the computer as a visual arts medium;
- To use artificial intelligence techniques in the context of visual arts;
- To use real time three-dimensionality computer driven imagery;
- To create an environment where visual shapes and individual characteristics evolve in time.
- And finally, to explore the irony of constructing a self-sufficient utopian world.

In this paper, I will present an accurate description of this world, which will be structured in two parts: a descriptive one, where I will present the creatures and their behavior; and a technical one, where I will explain their design and implementation. Secondly I will present a short overview of related work from different authors/artists. And finally, I will present the considerations and achievements of this research.

To fulfill these premises, xTNZ is a "virtual world" using 3D techniques where the living creatures evolve their shapes over time by mutant reproductions (Plants), or have implemented artificial intelligence algorithms in their behaviors (Birds).

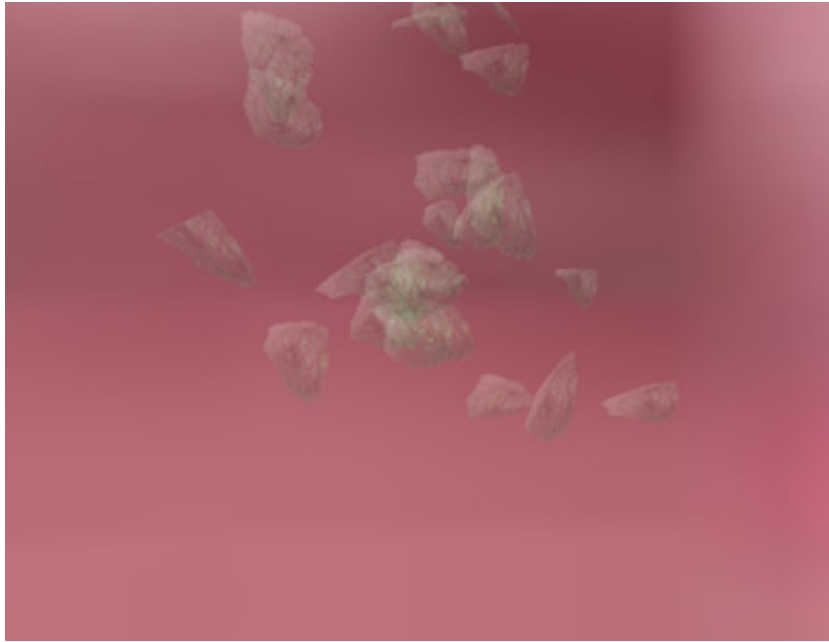


Figure 1: A cloud of particles.

2 Description of the virtual reality system

In generic terms, xTNZ can be described as an ecosystem. Three planets hold a community formed by two families of birds and two families of plants. Perpetual rain energizes the soil from which the plants feed. Birds fly around each planet foraging for food from plants. This balance can only be disrupted by the user/viewer and his/her presence; or the lack of memory on the system. The creatures have a natural life cycle; they feed, reproduce and die. From these basic rules, and the interaction of their behaviors, complex social environments can emerge.

2.1 The world entities and rules:

2.1.1 *Seasons/Time:*

The main system regulators are the days and the seasons. The processes of growth and reproduction are triggered according to these regulators. The system unit is a day, which is a configurable number of milliseconds.

2.1.2 *User action:*

The user interacts with the world via the standard keyboard and mouse input and the screen as output. The user navigates through zTNZ as if flying through a dream. In addition, the actions of the user in the world influence the ecosystem. Creatures will not reproduce when sensing its presence (with keyboard movements); thus the user wounds the overall balance of the ecosystem. The user also plays a role in the soundscape. Each moving creature has a sound, which intensifies or decreases depending on the distance to the viewer.

2.1.3 *The Clouds:*

Clouds are groups of moving particles flying over the planets. The function of the clouds function is to energize the soil. According to seasons they are more or less energetically charged. Their movements are continuous but chaotic along one planet. Therefore, they each follow a path, which is randomized but considers the previous movements and the wind direction.



Figure 2: View of a planet surface.

2.1.4 *The Land:*

The three planets from xTNZ support the ecosystem. Clouds or dead plants energize the soil. The land functions as the place where plants locate and as a source of food. Its texture is almost transparent and changes according to the viewer position.

2.1.5 *The Plants:*

Two families of plants inhabit this world: *Trolarindos* and *Imbuguzus*. They are both from the same family, thus they have the same basic rules:

- Both feed from the soil where they are located;
- Both feed birds with their energy when birds come to eat;
- Both have a variable life expectancy, which is defined by their previous generations.

If plants are mature during reproduction time (according to the season and days in the season), they can reproduce with other mature plants in their boundaries. When reproducing, the newborn inherits the parent's blended shape. In the last ten years of life they can reproduce with other species to produce mutations.

2.1.6 *The Birds:*

Two families of birds cohabit on xTNZ: *Crystals* and *Tiroliros*. All of them are a-sexual and reproduce once they are mature. As individuals they have a spherical shape. However, the way mothers aggregate their children varies between each species. *Tiroliro* mothers connect their children one after the other in a continuous shape. This branch bends when the mother moves. On the contrary, *Crystal* mothers aggregate their children to themselves in a bumpy shape. When reproducing, crossed reproduction between the families can occur producing mutant birds. When there are more mutants than original members of the family, a family mutation occurs blending the textures and the sounds. *Crystals* also rapidly increase and decrease in size during feeding. Each of these families has a common sound. Each individual emits a sound which is intense as the distance to the viewer augments.

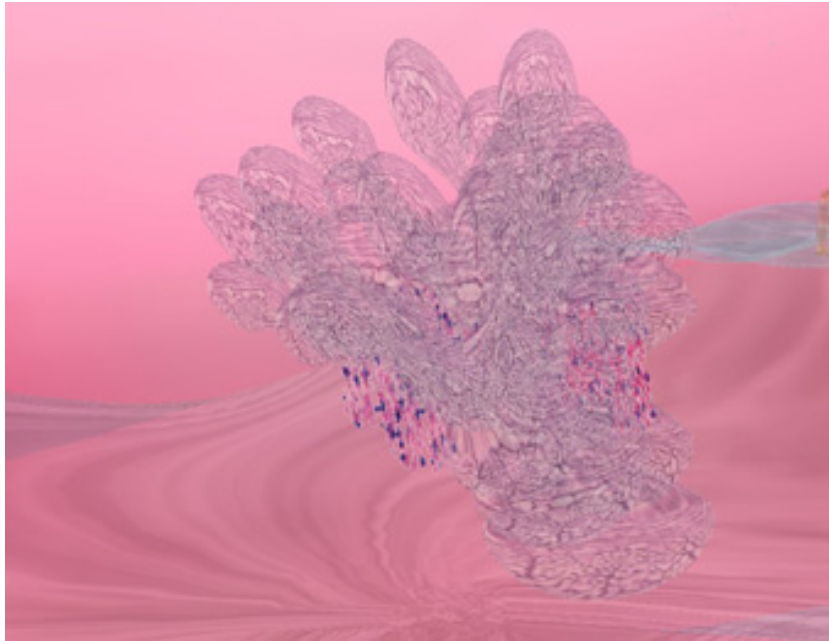


Figure 3: View of a mature Imbuguzu.

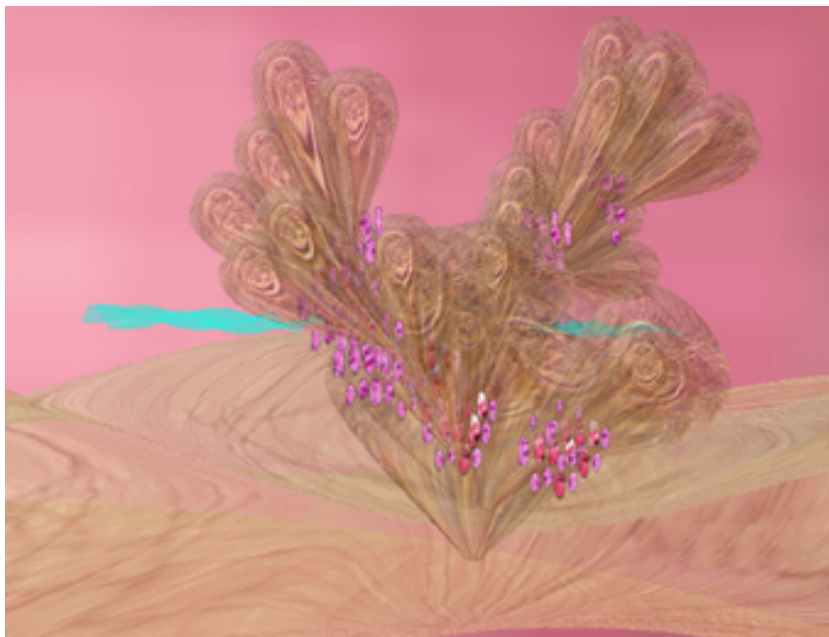


Figure 4: View of a mature Trolarindo.

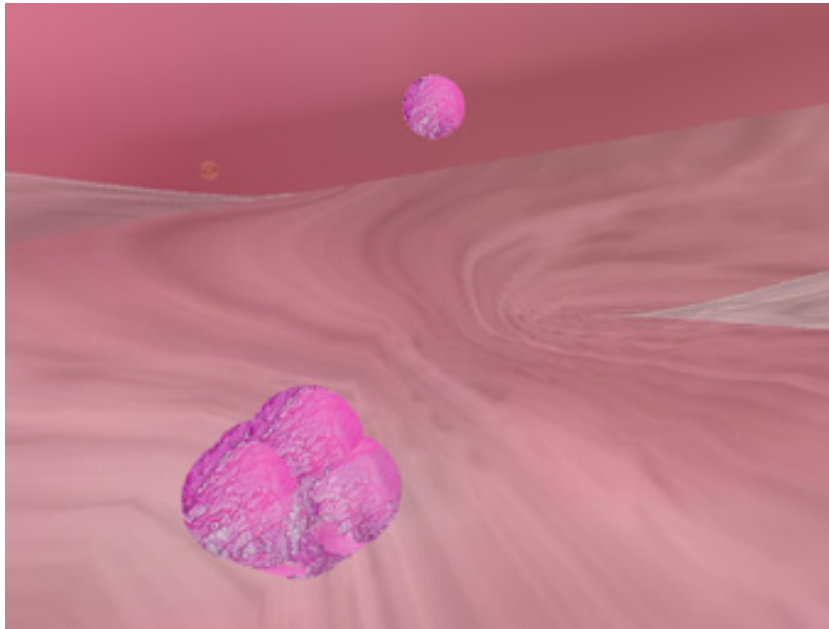


Figure 5: View of Crystals mother carrying its offspring and an single individual.



Figure 6: View of a Tiroliro mother with its offspring.

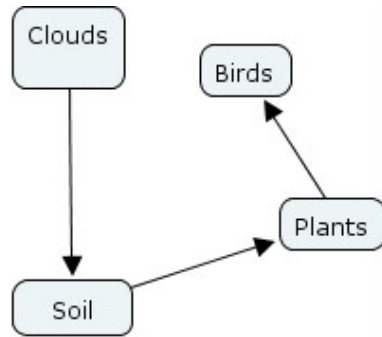


Figure 7: Graphical representation of the feeding chain.

But as they are both originate from the same family they have the same basic rules:

- They fly all along their planet in flocks. If the viewer is too close they escape from him;
- When hungry, which means below an energy limit, they fly to the nearest plant to feed from its energy; when eating, they stand in the ground next to the plant;
- During reproductive times, mature Birds mate with the nearest mature Birds;
- Mothers carry their children attached to them, creating a common shape. Only when the mother dies are the children released and able to start acting as individuals;
- Their life expentancy are pre-set and inherited form their parents;
- In the last ten years of life, Birds can reproduce with other families of birds to produce mutants.

2.2 The feeding chain:

The perpetual rain energizes the soil. Plants feed from the soil’s energy and birds feed from the plants.

3 The system technical implementation

In this section, will be outlined the implementation of the system. It will be explained the architecture design choices and the coding structure.

In order to fulfill the previously stated aesthetic requirements for this project, a real-time 3D engine was necessary. I have chosen Java3D since it allows a 100 percent usage of this language. Java3D uses a higher level of abstraction from the mechanics of rendering, allowing the programmer to concentrate on what is shown rather than how it is rendered. It also has a higher-level scene description, the scenegraph that allows scenes to be easily described, transformed, and reused. To develop the code for the project, I have used the free Eclipse package from www.eclipse.org, the Java 1.5.0 Software development Kit from <http://java.sun.com/javase/downloads/index.jsp>, and Java3d 1.4.0 <http://java3d.j3d.org/download.html> from Sun.

As presented in the “Description section” there are six entities in the world: clouds, land, plants, birds, planets and users. All these agents are created by instantiating the family classes who later deploy the individual creatures. A design compromise was made between performance and aesthetics. In this project one of the biggest issues in performance is the memory required. In order to reduce the memory needed, the creatures share their sounds ans appearances with all the members in their families instead of having their own individualities. To facilitate the programming work and readability, the code was structured in several family packages, as we will see in the next subsections.

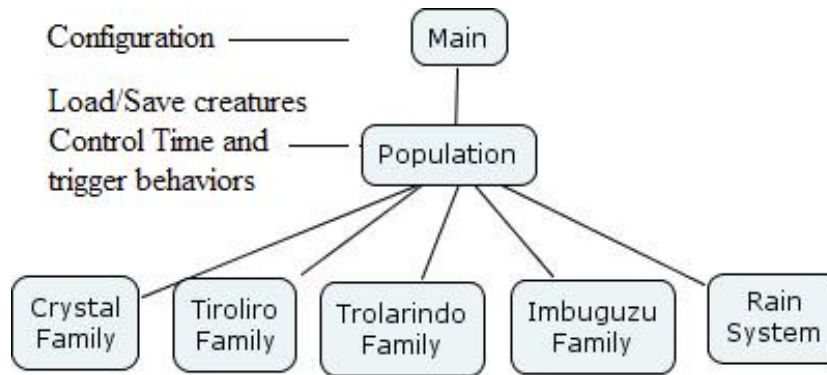


Figure 8: URL model of the core modules on the system.

3.1 The core classes - The package Global

This package stores the core modules of the system. The modules responsible for setting the configurations, filter the user input, create the creatures, and trigger the events during execution.

3.1.1 The MainClass class

MainClass is responsible for all the configurations required to set the three-dimensional system.

The *scenegraph* is a hierarchical data structure that captures the elements of a spatial relationship between objects. There are two distinct branches within the scenegraph: scene and view. The scene branch is where the relation of objects are defined. The view branch contains a ViewPlatform node and defines scaling, rotation and translation information. The view branch is responsible for rendering the scene branch to its attached Canvas3D component[J3DSpecification].

MainClass is where this entire system configuration takes place. The system starts with the configuration of the visual interface, declaring a Canvas3d and a SimpleUniverse.

A BranchGroup connects the scene elements to a TransformGroup, allowing the movements of the viewing branch. The static scene is composed by the Background, the Light and the Fog; the view elements are connected to each other by the TransformGroup that carries the movements of the viewing platform. A BranchGroup attaches both the static and the non-static elements to the SimpleUniverse locale.

The Mainclass is also responsible for the instantiation of the interface classes MouseBehavior and MyKey-NavigatorBehavior. Also populates the world declaring a new Population object. Population is the class responsible for the creation of the creatures and controlling the system time.

3.1.2 The Population class

Population keeps two updated ArrayLists of birds and plants.

The constructor initializes calling the method LoadData. This loads the creatures at the stages they were saved the last time the system was shut down. For each family of creatures, LoadData initializes the proper family classes and the proper list of creatures. Then, depending on the kind of creature, it calls the specific database file to load them. For each loaded creature from the saved file, this database class calls the family method LoadNew(Creature) to add it to the scenegraph.

Every time this module is triggered (a system day), the run method is invoked and specific behaviors are prompted according to the season and the day in the season. In all cycles, a memory check is made in order to avoid a system collapse due to running out of memory. If there is no memory, no reproduction will occur and the garbage collector is invoked. However, all the growing behaviors are activated even if there is no memory available.

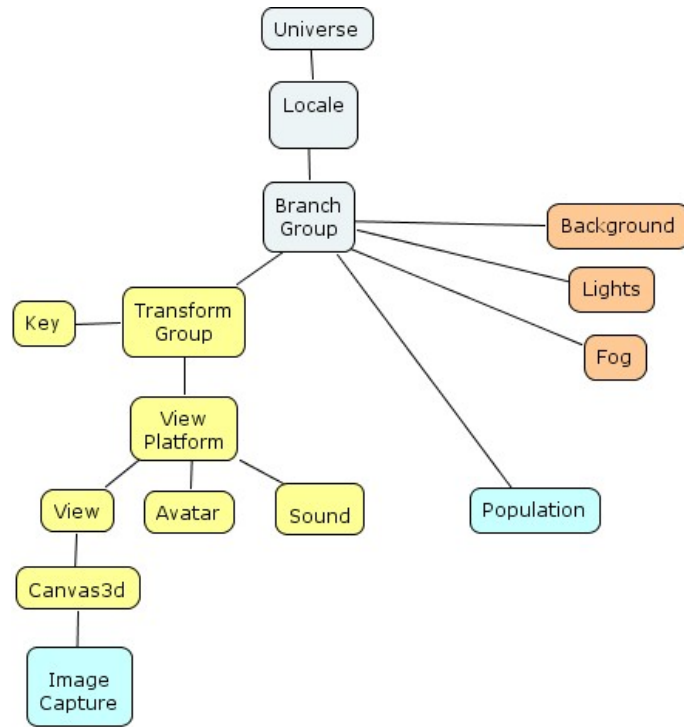


Figure 9: The Scenegraph structure.

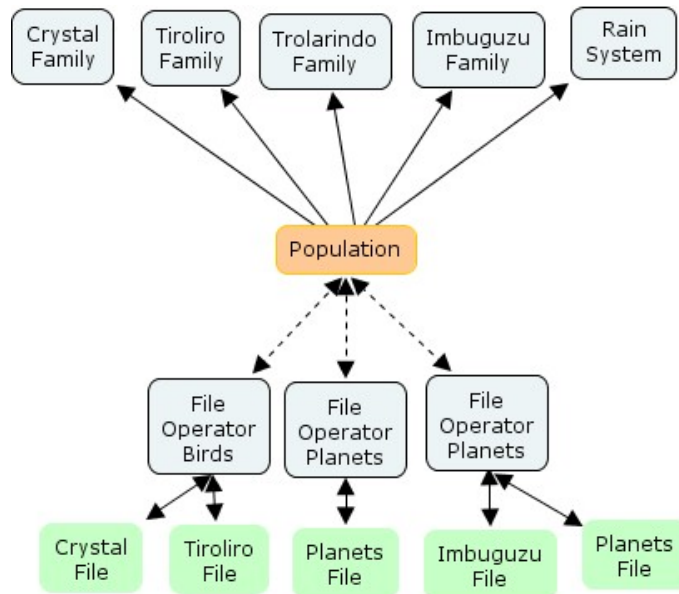


Figure 10: The system core modules.

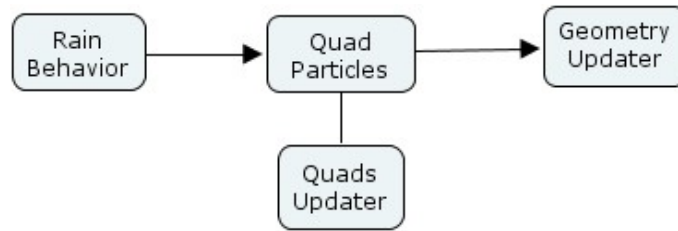


Figure 11: UML model of the raining system.

3.1.3 The MyKeyNavigatorBehavior class

This class extends Behavior and filters the keyboard, inspecting the AWT events. According to the pressed keys, a corresponding action will occur, such as creating new creatures, resetting the system, or moving the ViewPlatform transformGroup.

3.2 The raining system - package Clouds

The implementation of the raining system uses the particle system technique. William T. Reeves first introduced particle systems in 1983 [Reeves]. Particle systems are a powerful technique used for modelling natural phenomena. Individual particles are controlled by numerical values representing their position, size, velocity, colour, etc. In this way, is possible to mimic dynamic effects such as rain or water. Since they both describe fountain like examples, for this module I have based the code on the chapter “Interaction and Animation” from the Java3d tutorial [Tutorial] and “Particle Systems” chapter from Andrew Davidson book [Davidson]. The RainBehavior is the family class responsible for creating the new clouds.

3.2.1 The RainBehavior class

RainBehavior extends Behavior, to make the movements of the clouds over time. This is an independent thread (instead of using the time events in Population) to produce a continuous visual flow on the cloud movement.

This class is responsible for keep updated an ArrayList with the living clouds in this planet. Every cycle of time, RainBehavior is triggered and makes the movement methods in each instance in the ArrayList. A common Appearance is generated and stored here on this class.

When instantiating RainBehavior (the family class) a thread is thrown to the system and a specified number of QuadParticles (clouds) are instantiated.

3.2.2 The QuadParticle class

Each cloud is composed by a specified number of quadrilaterals (“quads”). A subclass of GeometryArray, QuadArray stores the “quads” of the cloud. Each QuadParticle has an inner class, which extends the GeometryUpdater class. This is necessary to keep the quads (the particles) updated at run-time. Every cycle, RunBehavior invokes the updateGeometry method in QuadsUpdater, which is then responsible for updating the geometry of the QuadArray with the renewed positions of the quads. The geometries are defined inside an OrientedShape3D so that the quads automatically turn to face the viewer, and can be set and rotated about a particular point or axis. Each cloud has a TransformGroup defining its position. Its value is kept in the vector *direccao*. The method moveFlower updates this vector, considering the wind direction, the previous three directions, and a random factor.

3.3 The planets - package Soil -

This package only contains one class, Soil. This class builds the three planets. Only the creation formula differs from each other.

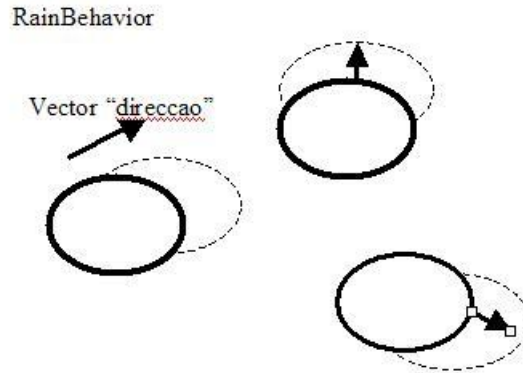


Figure 12: The vector *direccao* determines the future position of the cloud.

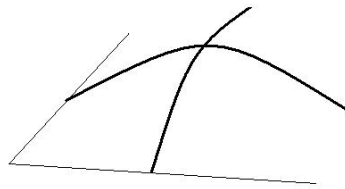


Figure 13: The surface of hte landscape is produced using the mathematical functions cosine,sine and tangent.

3.3.1 The soil class

The soil class defines a surface and implements a dynamic texture to it. This class primarily responsible for an keeping updated a matrix of energetic spots coinciding with its surface. Its visual representation was based on the chapter “Using Texture Images” from Daniel Selman’s manual [Selman]. The constructor receives the scene Branchgroup and calls the method createBranchGroup to create its own BranchGroup. This design strategy is meant for future developments of translation or any other transform operations we want to make in the future.

The method createDemLandscape creates the surface coordinates (stored in a pointArray). It defines the coordinates by pairs of four (a quad). The height is defined by the application of mathematical forms using the sine, and cosine functions to produce undulating surfaces, as well as the tangent function to produce picks on it.

The texture was mapped using SPHERE_MAP to produce the effect of an object reflecting its surroundings. This produces a dynamic effect of a texture changing according to the viewer position. A 50 per 50 matrix of energy is created and this class has the methods to keep it updated. The matrix coincides in space (x,z) with the defined quads of the surface.

Other techniques I have chosen the use of this technique of terrain representation instead of the popular *Fractal surfaces*. Fractal surfaces are meshes created using recursive techniques to model the height of the surface vertex surfaces⁹. Despite the naturalistic effect from this technique, my approach allows more abstraction and is far more visually appealing.

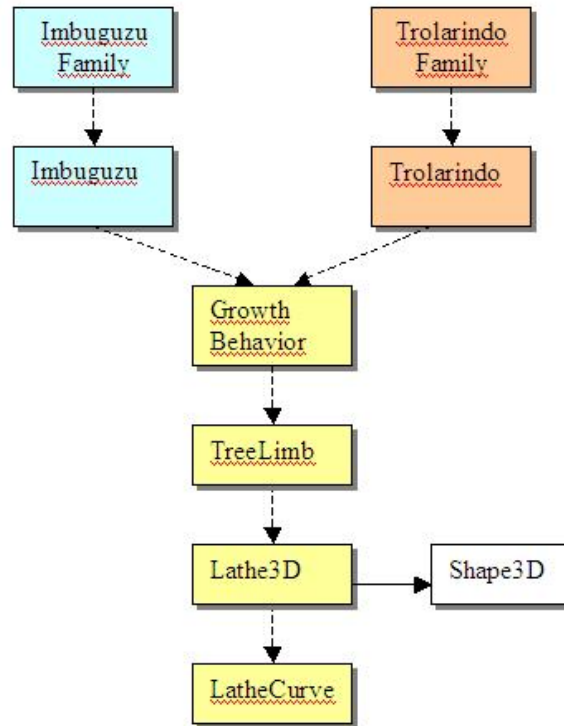


Figure 14: UML model of the plants.

3.4 The plants - package Plants

To create objects, Java3d uses geometry arrays to store the vertex or geometric primitives such as box, cone, cylinder and sphere. The chapter “Using a Lathe to Make Shapes” from Andrew Davidson [Davidson], describes the implementation of Lathe3d, a class that allows the creation of smooth objects. From the same book, the chapters “Trees that grow” and Scott Teresi’s “Modeling trees” <http://teresi.us/html/main/programming.html#trees>, they both follow the technique of dividing the branches of a tree in small cylinders, that can be grown independently of each other. Since it simulates a botanic growing procedure, I have adopted this technique to apply here. To produce mixed shapes, a minimum of two species is needed. In this system, plants are divided in two families: Imbuguzus and Trolarindos. Both are from the same species thus sharing the basic classes of growing and visual representation.

The plants’ visual growth is independent from the effective system time, since it occurs in separated modules. The Population object grows the internal data according to the system time; this class also triggers the reproduction behavior. A different thread rules the visual growth. Each plant has its own thread in order to grow individually at different times.

3.4.1 The GrowthBehavior class

This thread class, GrowthBehavior, is common to all plants. It extends Behavior and processes “genetic” information that is passed and mixed through the generations. Every time this thread is activated, after a test to the memory (only if there is enough memory available it will grow visually) the ApplyLimbs growing procedure is called to apply the growing rules to all the limbs in the limbs ArrayList. Each limb is individually grown so each one has some particular differences to the others. The growing rules are based on the limb’s age, size, and relative position on the tree. Some randomness exists in the growing direction.

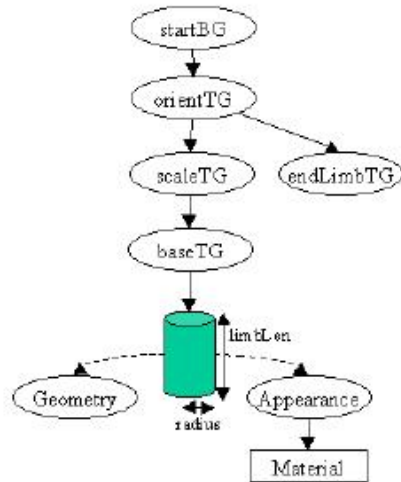


Figure 15: The TreeLimb scenegraph[Davidson].

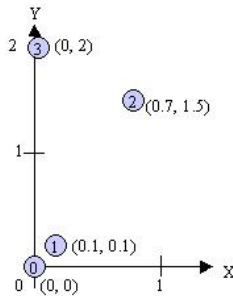


Figure 16: Two coordinate arrays defines the curvature of the shape.

3.4.2 The TreeLimb class

The TreeLimb class produces a limb. TreeLimb class has the methods for implementing the growth of the branch, making it grow longer or thicker through a non-uniform scale. To apply these transformations, each limb has three TransformGroups: one for the orientation; another for scaling operations, and a last for the object.'s position.

The ending object in this structure is an instance of Lathe3D, representing the geometry of the limb.

3.4.3 The Lathe3d class

Lathe3d is a subclass of Shape3d. The constructor produces an object in two steps. First a LatheCurve produces the curved shape and then this shape is rotated On the y-axis to produce a tree-dimensional geometry.

The method createGeometry passes the produced lathe curve coordinates to surfaceRevolve, which returns the coordinates of the resulting shape. These coordinates are used to define a QuadArray. The geometry is completed with vertex normals (to reflect light) and the texture coordinates.

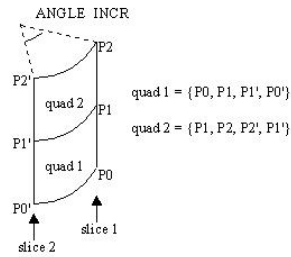


Figure 17: The curve is rotated defining a QuadArray geometry.

3.4.4 The LatheCurve class

LatheCurve receives two arrays of coordinates and produces a new list with interpolated points to represent the curve. It uses hermite curves to produce the curvature. The methods to calculate the interpolation points are described in Davidson Book with great detail.

3.4.5 The ImbuguzuFamily

This class has an Arraylist with all the living Imbuguzus. It contains the methods common to the Imbuguzu species, such as growing, reproduction or dying, as well as the common Appearance. The constructor instantiates five new Imbuguzus or loads Imbuguzus from the file.

3.4.6 The Imbuguzu class

This class contains all the “genetic” information. Different constructors are used for the three different situations of creation: normal reproduction, reproduction forced by the user, or mixed reproduction. When an Imbuguzu is created, a new TreeLimb is instantiated and passed to a GrowthBehavior object. This behaved instance is attached to the Imbuguzu branchgroup.

3.4.7 The TrolarindoFamily

This class has an Arraylist with all the living Trolarindos. It contains the methods common to the Trolarindo species such as growing, reproducing or dying, as well as the common Appearance to all family individuals.

The constructor instantiates five new Trolarindo or loads Trolarindos from the file depending on if it is a load operation or a user forced creation.

3.4.8 The Trolarindo class

This class contains all the “genetic” information. Different constructors are used for the situations of creation: normal reproduction, reproduction forced by the user, or mixed reproduction. When a Trolarindo is created, a new TreeLimb is instantiated and passed to a GrowthBehavior object. This behavioral instance is attached to the Trolarindo branchgroup.

Other techniques: The representation of Trees is a current hot topic in computer graphics according to Tim Dapper, from the Bremen University, Trees www.td-grafik.de/artic/talk20030122/overview.html. Plant modeling methods can be procedural, rule-based or hybrid. Procedural methods, such as the implementation used by SpeedTree, are used in the games industry due to the realistic results. However, these methods are not free. Rule based methods such as L-Systems¹⁰ are both popular and produce good results. Nevertheless, this is not a good solution to implement in a real-time system since L-Systems sees growth of a tree as a new tree completely replacing the old one. This has vital consequences when implementing growth in Java 3D. The natural approach, and the most disastrous from an efficiency point of view, is to discard the current tree at the start of a rewrite and generate a new one matching the new string expansion.

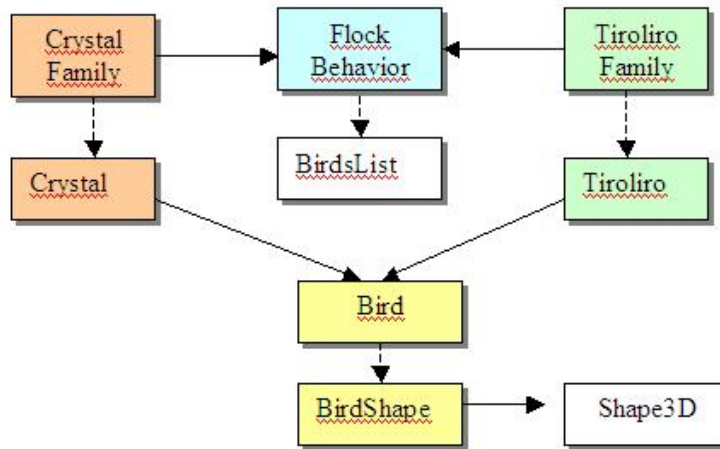


Figure 18: The birds UML diagram.

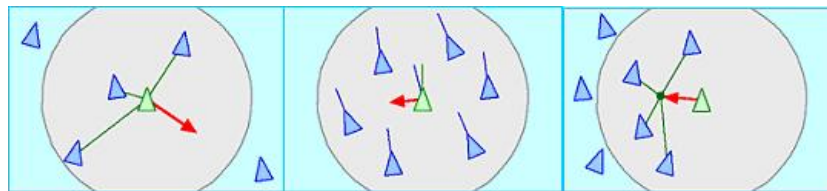


Figure 19: Reynolds steering rules: Separation; Alignment; and Cohesion

3.5 The birds - package Birds -

As I already mentioned, birds are divided in two families: Crystals and Tiroliros. This way they can mix their “genetic” information. But both are from the same species, so functionally, they share the basic structure of classes. On the implementation of the flying and flocking behavior, I have used an artificial intelligence technique called Flocking ¹¹. It autonomizes the movement and introduces a credible living effect on the birds’ behavior. Craig Reynolds first introduced flocking in 1987 [Reynolds], and this basic model became a standard replicated in several artificial life systems, such as the games Unreal or Half-Life [Davidson]. The basic flocking model consists of three simple steering behaviors:

1. Separation: steer to avoid crowding local flock mates.
2. Alignment: steer towards the average heading of local flock mates.
3. Cohesion: steer to move towards the average position of local flock mates.

I have implemented a version of Reynolds algorithm based on Andrew Davison’s “Flocking Boids” chapter [Davidson]. In this implementation, a thread runs each family independently. Every few milliseconds the procedure is triggered to update the position of all elements in the family: In other words, a vector is updated with each of the three flocking rules. Next, the bird position is updated by applying this resulting vector to the specific TransformGroup. In the same way, this vector is updated with the inverse position of the viewer when he is too close to a bird in order to make the bird run away in the opposite direction. Specific behaviors of each species (for instance, the way they aggregate to the mother) are defined in the lower-level specific classes. However, the birds share two lower level classes. These classes define what is a bird, both visually and as a data structure. This is the information that is mutated on the crossed reproduction (for instance, the velocity or the perching time) The vector, birdPos in Birds is the key to the bird’s movement. The birds’ class and the specific Tiroliro and Crystal’s classes have the rules to fly, updating this vector. However, FlockBehavior carries the flocking rules.

3.5.1 The FlockBehavior class.

CrystalFamily or TiroliroFamily extends this class. This class creates the list of birds when instantiating BirdsList, but its main function is to implement the movement of the birds' flight. Every cycle of time the Bird class method animateBird is applied to all the elements in the Birdlist to move the bird.

This class defines the rules of flocking with the methods cohesion, separation, and alignment. All these methods return vectors, which are then used by the subclasses to update the bird position.

3.5.2 The TiroliroFamily class

This class extends FlockBehavior. The constructor immediately calls the FlockBehavior constructor in order to have a list available to create the birds. These are created calling createBirds, which instantiates the specified number of TiroliroBird objects. All the specific behaviors of Tiroliros are specified here, such as finding the closest plant, the growing procedure, the reproduction procedure, and the death function. TiroliroFamily also creates the Appearances, which will be referenced by the Birds objects.

3.5.3 The Tiroliro class

This class extends Bird. Its function is to keep the movement rules specific to Tiroliros. In the method appendSibling, it also defines method to attach newborn Tiroliros to their mothers in the reproduction time. The animateBird method is overridden to implement the "shaking" movement of the Tiroliro's offspring.

3.5.4 The CrystalFamily class

This class also extends FlockBehavior. The constructor immediately calls the FlockBehavior constructor, in order to have a list available to create the birds. These are created calling createBirds, which instantiates the specified number of CrystalBird objects. All the specific behaviors of Crystals are specified here, such as finding the closest plant, the growing procedure, the reproduction procedure, and the death function. CrystalFamily also creates the Appearances, which will be referenced by the Bird's objects.

3.5.5 The Crystal class

This class extends Bird. Its function is to keep the movement rules specific to Crystal. In the method, appendSibling, it also defines the method to attach newborn Crystals to their mothers in the reproduction time. The animateBird method is overridden to implement the "scaling" behaviour when the Crystal bird is eating.

3.5.6 The Bird class

The Bird class constructor initializes the variables, creates a new BirdShape object and initiates the sound.

FlockBehavior calls the animateBird method every 100 ms. This method controls most of the bird behaviors, such as the perching time, keeping the movement inside the world boundaries, and setting the new position of the positional vector birdPos. This is updated in calNewVel by the application of the three Reynold's flocking rules from FlockingBehavior and the inverse position of the viewer when he is too close to the bird.

3.5.7 The BirdShape class

Birdshape extends shape3d andbdefines the geometry of the bird. The constructor creates a single ball. The method appendBalls returns a BranchGroup with a ball attached to a cylinder in different configurations according to the specified parameter input.

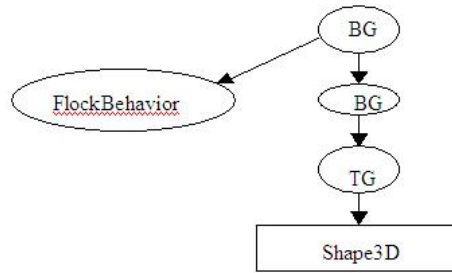


Figure 20: The Bird’s class scenegraph.

3.5.8 The BlendBirds class

This class offers the functions for the mutation procedure. If there are a bigger number of new creatures than the current generation members, a mutation will occur. `blendBirds` method, called in `Population`, triggers the `blendImages` and `blendSounds` functions, when this happens. The first method, `blendImages`, combines the texture images from the current generation of the two mixing species. The first one will be stronger, so the final image is the result of the application of the formula $3/4 * \text{Img1}[n] + 1/4 \text{Img2}[n]$ convoluted to the image. To do so, the two `jpg` images are converted into `bitmap` files. The resulting image is then converted to `jpg` again. On this operation the classes The first family of birds will apply this image file as texture to all the members of the family.

The mutated sound is produced in `blendSounds`. This method reads both input files sequentially, and combines them, by selecting the bigger from the compared bytes as the output one. The resulting file is then applied to the family members.

Other techniques I have considered the utilization of metaballs since the spherical shape of Birds is ideal for its application¹². However, despite all its visual appeal, its implementation is too slow to produce in real time 3D. A truly rough version of it was implemented instead by using cylinders to connect the spheres.

3.6 The file system - package Database

To preserve the data image when the system is shut down, save and load operations were implemented on this package. To accomplish this, a model/view paradigm of software architecture was applied, so the creatures have both a visual and a data representation permanently updated. The data is loaded/saved into binary files located in the local disk. All these methods are called in `Population` on `MainClass`.

3.6.1 The FileOperator class

This class carries the methods to open and to close the files. This class is to be extended since all the other classes of this package use these functions.

3.6.2 The FileOperatorPlanets class

This class extends `FileOperator` and uses buffers to carry the information from the memory to the files and *vice-versa*. When loading from the file, for each record, calls the `Population` method `LoadNewPlanet` to create new objects on the scene. For ease of use, an inner class was used to redefine the disperse data as a single “record” object.

3.6.3 The FileOperatorBirds class

This class extends `FileOperator`. It defines a buffer and uses it to carry the information from the memory to the files and *vice-versa*. When loading from the file, for each record, calls the `Population` method `LoanNewBird` to create new objects on the scene. For ease of use, an inner class was used to redefine the disperse

data as a single “record” object. Since different families of birds exist and the essential data to preserve is the same, each family has an assigned code to distinguish between them (1- Crystals 2-Tiroliros).

3.6.4 The FileOperatorPlants class

This class extends FileOperator. It defines a buffer, and uses it to carry the information from the memory to the files and *vice-versa*. When loading from the file for each record, it calls the Population method LoanNewPlant to create new objects on the scene. For ease of use, an inner class was used to redefine the disperse data as a single “record” object. Since different families of plants exist and the essential data to preserve is the same, each family has an assigned code to distinguish between them (1- Trolarindo 2-Imbuguzu).

3.7 The image capture - package Media

Since one purpose of this work is the production of landscaped photo-like images, I have implemented a capture system, both to still jpeg images and mov Quicktime video. Both are activated through the keyboard.

3.7.1 The CapturingCanvas3D class

This class creates jpeg images from a Canvas3d image shot. This class was implemented by Peter Kunstz <http://java3d.j3d.org/faq/capturing.html>. It extends Canvas3d by overwriting its postSwap method. In this method it creates a raster, which is then passed to a buffered stream file that is further converted to the jpeg format.

3.7.2 The CapturingCanvas3D2 and the other movie capture classes

Inspired by the previous class, T.E.A de Souza created this version for movies. Once again, the Canvas3d is extended to overwrite the postSwap method. However, instead of saving this single image, it creates an MovieBuilder object. This interface activates the QTMovieBuilder, which implements the Runnable class so it runs in parallel with the system. It captures all the screens until stopCapture is called (via the keyboard). Next, this movie is transformed into the QuickTime format using JPEGImagesToMovie. Since Quicktime uses jpeg compression, no format conversion is needed.

3.8 The sound system

The Java3d API has some helpful tools to work with sound. It works based on bounding areas attached to the objects. Each object has a trigger area and different levels of intensity can be described to different triggering distances. This allows the system to render the sound according to the distance to the viewer. Also, the API has methods to define a background sound. All these functionalities allow the creation of a soundscape enveloping the visual representation of xTNZ. As sound files use a great amount of the memory, the same sound is shared by all the elements of a family. For instance, all the Tiroliros reference one common sound file instead of having their own private individual sound.

Now, and before I evaluate the system, and in order to make a contextual analysis of this project, I will review related art works[virtual art].

4 Background of artists using virtual reality

4.1 Three-dimensional synthesized images

One of the pioneers of using computer synthesized three-dimensional images with an artistic purpose is Yoishiro Kawagushi. He has been producing movies exploring all the appealing qualities of three-dimensionality

since at least 1987. His process is based on non real-time and high quality rendering to produce the videos. On his famous work "Embryo", produced in 1988, marine-like creatures mimic aquatic creatures' movements.

A recent important author is Jon Mac Cormack. In his 2001 piece "Bloom", he continued his investigation into computer synthesized natural spaces. Using his own advanced L-System algorithm, he produced images of mutated and crossbreed representations of native Australian flora. This work was presented to the public in large-scale prints. His proposition is that synthesized natures are becoming replacements for real nature in urban environments.

In a different style, Jack Shaw is considered to have introduced three-dimensionality into the visual arts through his installations. "Legible city" from 1990 is still referenced today as the primary seminal artwork to have pioneered the artistic appropriation of the three-dimensionality [Popper, Christiane]. In this piece, Central Manhattan is mapped in words and sentences, which are based on a particular journey that the viewer experiences on a bicycle interface.

In my opinion, the work to have the strongest impact on viewers is Charlotte Davies "Osmose". A full body experience was successfully achieved using immersive technologies. Participants navigate through the space (a visually powerful real-time simulation of nature and text zones seen on a HMD), using a chest vest containing devices sensitive to the body's breathing. Just as when diving, when the viewer's lungs fill with air the viewer rises. In 1998, whilst continuing to explore immersive environments using HMD, Davies created "Ephémère", A virtual space that generates reactive image worlds in real time. Structured in three-levels, landscape, earth and body, it includes organs of the body, bones and the circulatory system, in a symbolic correspondence between the interior of the body and the subterranean earth.

4.2 Evolving environments

Since the 70's, "artificial-life environments" were introduced and became part of the computer sciences. John Conway's "Game of Life" is one of these first genetically evolving environments. A collection of cells, based on a set of rules, can live, die or multiply. Depending on the initial conditions, the cells form various patterns throughout the course of the game. In artificial life environments, using genetic algorithms, virtual creatures are created and exist within virtual environments. Artists soon adopted this language to explore their concerns with the creation of life processes in digital media - life that interacts with its environment and evolves.

Karl Sims, a biologist/programmer, recalls in his work the Darwin's natural selection theory. In "Evolved Virtual Creatures", a population of cubes performs tasks (such as running, fighting or swimming). Following these tasks, those cubes that are the most successful survive and their virtual genes are replicated to control the cube in which they live. Created in 1995, "Galapagos" is an installation where a computer generates twelve different organisms at every iteration. Visitors have to select the most "aesthetic" organism that will continue to live, reproduce and mutate.

The mutation of forms is also the interest of William Latham's work. He has been producing organic-like forms based on natural geometry with his program, "Mutator", since 1987. The program generates an offspring of mutated forms. Next, the author makes an "aesthetic" decision selecting the forms of the genes that will become the seed for the next generation. The creatures are finally presented as still images or movies.

Jane Prophet's work on "TechnoSphere" developed a different approach by using configurable forms. Users can create artificial life forms by freely combining a number of "body parts" that are offered on the site. Next, the creature is placed in a virtual landscaped environment. "TechnoSphere" is a virtual space where digital life forms compete for food and develop their own genetic code to survive. The evolutionary development of this 3D world is dependent on the participation of an on-line audience, and the creatures send e-mail messages to their creators updating them with their status.

At the same time as Prophet's work, Mac Cormack created "Eden". A world populated by collections of evolving virtual cells. The cellular creatures move in the environment by making and listening to sounds, foraging for food, encountering predators and possibly mating with each other. Over time, creatures evolve to fit their landscape. A simple physics dictates only three basic types of matter in the Eden world: rocks, biomass and sonic animals.

Finnaly, Christa Sommerer and Laurent Mignonneau had interesting approaches of their own. "Life

Spacies“ is a virtual world where virtual organisms interact with each other, appear and evolve in response to the position, movement and interactions of visitors, as well as on-line visitors. In the famous 1995 work “A-Volve”, visitors interact using their hands with virtual creatures in the space of a water filled glass pool. These creatures are designed by the users and immediately start swimming. The fittest creature survived the longest and is able to mate and reproduce. The creatures compete by trying to get as much energy as possible.

Nevertheless, despite these approaches, evolutionary worlds have been primarily focused on scientific outputs in the fields of artificial intelligence and biology rather than art contexts. Polyworld www.beanblossom.in.us/larryy/PolyWorld.html, Nerve Garden www.biota.org/nervegarden/ and Animat Vision www.cs.ucla.edu/~dt/animat-vision/, are good examples of the scientific developments on this topic.

Commercial approaches of artificial life and evolution include areas such as computer games industry in which “Creatures” and virtual toys, such as “Tamagoshi” are suggestive examples [Grau].

Now we move to conclusion in light of what we have just discussed.

5 Conclusions and future developments

The focus of my work is mainly on aesthetic issues. What makes this project different from all the approaches I have discussed is that I am using artificial intelligence techniques and three-dimensionality to produce a generative program¹³. There are no concerns about accurately representing natural shapes or behaviors. There is a naive reference to it but simply in terms of evolution. However, the spotlight is on the abstract progress of the forms. Thus, not all forms evolve, nor do the sounds. Future developments will include this evolution. So, human images and sounds would be the input to generate something completely evolved by the computer.

To give to xTNZ the surprising effect of a renewed appearance every time it is visited, a strong factor of unpredictability was introduced. The evolution occurs based mostly on chance since the creatures don’t have the ability to learn. Nevertheless, the system navigation is not without its flaws. The joy of navigating through the space is degraded by the lack of novelty and the small extension of the world. Only five families of objects exist and only three small areas have “life” on them. Introducing some more species and planets could easily solve this novelty issue. Another issue is that on using only two families in each specie, the mutations will tend in time to be the average between the initial properties of two families. Introducing more families would solve this problem. However, in this phase of development, I have been more concerned on opening new possibilities to the future of the system than with creating a final version.

The navigation in xTNZ is severely disturbed by an analysis error that I made with regards to the lack of level of detail and mipmap structures. These techniques would have improved the performance significantly since the renderer and mostly the system memory would be relieved of many geometries and texture details. This must be the first improvement to be made in the future since this is a deep but necessary restructure of the system. Also, the impact of the Java Virtual Machine Garbage Collector (GC) is significant in the navigation progress. While GC is running, an appreciable system slowdown is noticed. The realism is lowered when several frames are dropped and the navigation becomes jumpy. Increasing the number of processors can attenuate this. Therefore, since the system is fully threaded, the performance would increase substantially.

Due to the aesthetic assumptions I have presented, another improvement has to be made. The viewer role changes radically due to another analysis limitation. The Java3D architecture is designed to render the scene when the viewer boundaries intersect the renderable region. So, instead of computing the entire scene, Java3D is only concerned with rendering the viewable area. This has some interesting consequences. One is that the user presence is necessary to trigger some visual growing procedures, such as the plants. Another is that bird’s movement are blocked by this lack of processing and as a result, birds die starving. This leads to a system design deadlock. To solve this problem the viewing area would have to be configured to a significantly larger size. Yet, due to the excessive memory and computing processing, this introduces a limit on the number of living creatures. The best solution is to change the 3d Behaviors of Java3d to standard Java Threads and control their life cycles.

Another future development for this project is to produce ray traced images. Since this will allow improvements on the quality of the final image with more accurate reflections and shadows, a high quality video can then be produced. However, Java3D is a real time renderer, and real time ray tracers, such as

OpenRT, are not yet available. Thus, one solution is to export the data model on every frame produced to a ray-tracer, such as POV-Ray.

6 Evaluation

This project is meant to be an artwork, but it is made in the context of a Masters in Science course, so it must be evaluated both artistically and scientifically. The evaluation of a project of a virtual world meant to be an artwork should focus on the aesthetic questions such a work raises as well as its pertinence to the visual arts contemporary context. If it is possible to define objective evaluation criteria for an artwork, in the current art context, the evaluation should focus on how the available techniques were applied to this specific case, on how technical solutions were applied whilst facing the artistic intent, and on how this has or has not led to a good result according to the standards of art criticism. However, this project is more than artistic research, and the scientific scope contextualizing this work must also be taken into account, such as the programming skills that were required, the correct application of the tools and scientific references, and the innovational view this work offers.

In terms of my own self-evaluation, I have considered whether or not I have correctly applied the new tools this Masters course has given me. I had previous experience with Visual Basic and C languages, but I only started using Java this year. With xTNZ, I have used Java3d on creating a self-generative 3D environment. This environment was populated with creatures modelled using 3d geometric structures, which progress in realtime. Some of this population (Birds) was animated applying flocking algorithms (Birds) and exhibits some emergent behaviors. Other programming techniques were applied such as the data maintenance; or the image and the sound manipulation (required in the mutation process). Also, with xTNZ, I have stated some ideas to the discussion on aesthetics about the computer status in society. Reviewing the premises stated in the introduction section, I can objectively say that all of them were successfully accomplished: the use of three-dimensionality, the introduction of artificial intelligence techniques, and the presence of an evolutionary behavior of the shapes according to interactions among the creatures.

However, xTNZ offers little in terms of scientific innovation and it needs further work in order to rectify certain fundamental problems, such as the fact that xTNZ cannot evolve completely on its own due to some flaws in the programming which make it crash sometimes. This inhibits, so far, the xTNZ exhibition on public for long periods (one day time for instance), since it requires someone to be nearby. Likewise, the project needs further work to resolve the navigational problems I have discussed earlier. In conclusion, despite the originality of this work (combining the techniques of artificial life with three dimensionality to produce a self generative programme, which is simultaneously a social critic) and the fact that the structure of the system is running, this project cannot be considered yet finished. It requires some restructuring and programming work to reach its full potential.

Acknowledgments

The development of this project was not possible without the help of T.E.A de Souza and his amazing programming skills, the provision of sounds by and amazing aesthetic discussions with Elena Gonzalez-Polledo, the editorial feedback of Joanna Pylak, as well as the help of my tutors, Professor Frederic Leymarie and Professor Mark d'Inverno. Finally I would like to warmly thank to all my beloved friends in Raymont Hall for their friendship and encouragement.

Notes

¹The name *xTNZ*, is an homage to *David Cronenberg* 's movie *Existenz* where he plays with the issue of reality and virtuality. It comes from the combination of the words *extension* and *existence* in the way they can be written on the new lexical format used on the domain of new technological chat environments such as SMS or MSN. Existence for the virtual life and identity the creatures on the world will live; Extension because this virtual world will be an extension of the natural world according to a set of rules that defines an ecosystem.

²For more information on Digital art see: [Popper] and [Christiane].

³I am using here the term *technique* referring the craft tool used to develop an artwork.

⁴"In 2008, video games industry is predicted to have more profits than Movies and Music industries altogether" in [Nelman]p.14.

For an historical overview of video games see [videoGames www.doc.gold.ac/~ma501ra/Videogames.doc](http://videoGames.www.doc.gold.ac/~ma501ra/Videogames.doc)

⁵"Empiricism presents the real in term of reality, a world differentiated into object and categories. Digital narratives that trade in empirical reality develop around several themes: naive, representational, scientific, mathematical, and materialistic forms of realism. ...

Empiricism presents the proximal as what is received through the senses. According to the empiricist, the real is defined as what is within our bodily grasp, through which the senses operate. Of course, our senses are not entirely reliable, and the reality and its appearance may be different. Locke distinguished between the primary or real qualities of objects, how they are in themselves and can be studied by science, and their secondary or apparent qualities, which is how they appear to the human senses. A further way of casting the distinction is between physical and sensible properties of objects. For example, there is the real color of an object, as determined by scientific measurements, and there is the sensible color, as we perceive it."

In [Coyne] , p.79-80.

⁶"Hans Moravec, head of the Carnegie-Mellon Mobile Robot Laboratory, in *Mind Children* argues that the age of protein-based life forms is drawing to a close, to be replaced by silicon-based life forms. Humans need not to despair, however, because they can have their consciousness downloaded into a computer. In the fantastic scenario in which he imagines this operation, Moravec has a robot surgeon cut away a human brain in a kind of cranial liposuction until all the information the brain contained is inside the computer and the skull is empty of brain tissue. Moravec reasons that once human consciousness is inside the computer, it is effectively immortal."

In [Moser/MacLeod]p.2.

⁷[Grau]

⁸"in computer culture it becomes common to construct a number of different interfaces to the same "content". For instance, the same data can be represented as a 2D graph or as an interactive navigable space. Or, a web site may guide the user to different versions of the site depending on the bandwidth of her Internet connection. Given these examples, we may be tempted to think of a new media artwork as also possessing two separate levels: content and interface. Thus the old dichotomies content-form and content-medium can be rewritten as content-interface. But postulating such an opposition assumes that that artwork's content is independent of its medium (in an art historical sense) or its code (in a semiotic sense). Situated in some idealized medium-free realm, content is assumed to exist before its material expression. These assumptions are correct in the case in the visualisation of quantified data; they also apply to classical art with its well-defined iconographic motive and representational conventions. But just as modern thinkers, from Whorf



Figure 21: Rewrite of F.

to Derrida, insisted on the “nontransparency of the code” idea, modern artists assumed that content and form cannot be separated. In fact, from the “abstraction” of the 1910s to the “process” of the 1960s, artists have continued to invent concepts and procedures to assure the impossibility of painting some pre-existent content.”

In [Manovich], p.66.

⁹“The plasma fractal uses a continuous mesh subdivision into smaller units. The mesh is first seeded with four corner points, and then a two-stage process is repeated until sufficient extra points have been created. In the first stage (the diamond step), the height of the midpoint of the four corner points is calculated by averaging their heights and adding a random displacement in the range $-dHeight/2$ to $dHeight/2$. Aside from the plasma fractal used in our code, two other popular approaches are the fault fractal and Fractal Brownian Motion (FBM). The fault fractal creates a height map by drawing a random line through a grid, and increasing the height values on one side of the line. If this is repeated several hundred times then a reasonable landscape appear. The main drawbacks are the lack of programmer control over the finished product, and the length of time required to produce a convincing geography. An FBM fractal is a combination of mathematical noise functions, which generate random height values within a certain range. An important advantage of FBM is that each noise function has several parameters, which can be adjusted to alter its effect. A crucial problem for an “industrial-strength” landscape representation is the enormous amount of memory required for a large map. For example, a height map made up of floats for (x,z) coordinates at 1mm intervals, covering a 10m square area, would require about 400 MB of RAM, and that’s before the cost of rendering and adding objects to the scene. The obvious answer is to reduce the sampling density, but this will also reduce the map’s resolution. A better solution is adaptive meshing, as typified by ROAM (Real-time Optimally Adapting Meshes). It is adaptive in the sense that low-resolution sub-meshes cover areas that are flat or distant from the viewpoint, while nearby or rocky areas use more detailed grids. Also, since viewpoints can move, the level of detail apparent at a particular terrain location will vary over time, as the user comes closer and moves away. ROAM creates its dynamic mesh with triangle bin trees (a close relative of the quadtree), which are split/merged to increase/decrease the resolution. Terrain creation is also used by simulation and GIS applications. A popular file format for geographic data is the Digital Elevation Model (DEM), which represents grids of regularly spaced elevation values.” In [Davidson] p.701-710.

¹⁰“Lindenmayer systems (L-systems) consist of rewrite rules, and have been widely used for plant modeling and simulation. Perhaps surprisingly, there is a direct mapping between the string expansions of the rule system and a visual representation of the plant. An example, using a bracketed L-system, will give an idea of how this works. The L-system contains a single start string "F", and rewrite rule: $F \rightarrow F [-F] F [+F]$ F The visual characterization is obtained by thinking of each 'F' as a 'limb' of the plant. The bracketed notation is viewed as a branch-creation operator; the '-' as a rotation to the right for the branch, '+' a left rotation. Consequently, the rewrite of 'F' to "F[-F]F[+F]F" can be seen as the plant expansion

Since each limb is an "F", rewriting can continue, creating longer strings, and more Complex plant-like shapes, as shown in Figure

There are a great variety of L-system languages, perhaps the richest being cpfg, available from <http://www.cpsc.ucalgary.ca/Research/bmv/>. It includes a full range of programming constructs, data structures, library functions, and various extensions such as parameterized L-system symbols, and sub L-systems.”



Figure 22: A sequence of rewrites.

In [Davidson] p.766-767.

¹¹"Particle Systems are an important component of many 3D games: when you see sparks flying, smoke swirling, fireworks exploding, snow falling, water shooting, or blood spurting, then it's probably being done with a particle system. A particle system consists of a large population of individual particles, perhaps tens or hundreds of thousands (although many commercial games use far fewer depending on the effect required). The particle system is in charge of creating and deleting particles, and updating their attributes over time. A particle is typically rendered as a graphics primitive, such as a point or line. This means that rendering overheads can be reduced, an important consideration when so many particles are involved. With the advent of more powerful graphics cards, simple polygons (e.g. triangles, quads) have also been used, which allows textures and lighting to be introduced. Particle attributes can be very varied, but typically include position, velocity, forces (e.g. gravity), age, colour/texture, shape, size, and transparency. Attribute updates tend to use time-based equations, but other approaches are possible. For instance, a particle's new position may be a random adjustment of its previous position. Particle systems often have a generation shape, which specifies a bounding volume in which particles can be created. Generation shapes have been extended to specify bounding volumes for particle updating and aging. For example, if a particle moves outside the space then it will begin to age, and age more quickly as it moves further away." In [Davidson]p.565.

For an historical overview of video games see [videoGames www.doc.gold.ac/~ma501ra/Videogames.doc](http://www.doc.gold.ac/~ma501ra/Videogames.doc)

¹²"A metaball is an isosurface in 3D space. Basically, define a function in 3D space, which takes as input the x,y,z coordinates of a point, and outputs a floating point value. Then decide on a threshold for this output; points in 3D space that fall below the threshold (when run through the function) are ONE, while points above the threshold are ZERO. This divides 3D space into two sets - think of it as solid (occupied) space versus empty space (air).

The continuous surface that is formed where the two meet (assuming the function's output is continuous/differentiable) is called an isosurface. This isosurface has a different look for each possible function."

In Ryan's Balls <http://www.geisswerks.com/ryan/BLOBS/blobs.html>

¹³"Generative art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art." In [Galanter]

References

[Popper] Franck Popper, "Art in the -electronic Age"; Thames & Hudson, 1993.

- [Christiane] Christiane Paul, "Digital Art"; Thames & Hudson, 2003.
- [Nelman] Nick Nelman, "Video Game Art"; Assouline Ed., 2005.
- [Coyne] Richard Coyne, "Technoromanticism - digital narrative, holism, and the romance of the real"; The MIT press, 2001.
- [Moser/MacLeod] Mary Anne Moser/Douglas MacLeod, "Immersed in technology - Art and virtual environments"; MIT Press, 1996.
- [Grau] Olivier Grau, "Virtual Art - from illusion to immersion -"; The MIT press, 2003.
- [Manovich] Lev Manovich, "The Language of New Media"; The MIT press, 2001.
- [Baudrillard] Jean Baudrillard, "Symbolic Exchange and Death", Sage, 1993.
- [Davidson] Andrew Davidson, "Killer Game Programming in Java"; O'Reilly, 2005.
- [Selman] Daniel Selman, "Java 3D Programming"; Manning, 2002.
- [Flanagan] David Flanagan, "Java in a Nutshell"; O'Reilly & Associates, Inc., 1996.
- [H.Deitel/P.Deitel] Harvey M. Deitel & Paul J. Deitel, "Java How to Program"; Prentice Hall, 2001.
- [Reeves] Reeves, W.T., "Particle System -a Technique for Modelling a Class of Fuzzy Objects"; Computer Graphics, 17(3), p.359-376, 1983.
- [Reeves/Blau] Reeves, W.T and Blau, R. "Aproximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems"; Computer Graphics, 19(3), p.313-322, 1985.
- [Reynolds] C. W Reynolds, "Flocks, Herd, and Schools: A Distributed Behavioral Model"; Computer Graphics, 21(4), SISGRAPH'87, pp.25-34, 1987.
- [Galanter] Philip Galanter, "What is Generative Art? Complexity Theory as a Context for Art Theory"; International Conference on Generative Art, 2003.
- [Vince] John Vince, "The language of Computer Graphics - A dictionary of terms and concepts -"; Architecture Design and Technology Press.
- [O'Regan/Noe] J.K. O'Regan and A. Noe, "A sensorimotor account of vision and visual consciousness". Behavioral and Brain Sciences, 24(5):883-917, 2001.
- [Varela/Thompson/Rosch] F. Varela, E. Thompson, and E. Rosch. "The Embodied Mind"; The MIT Press, 1991.
- [Tutorial] Sun Microsystems, "Java 3d API Tutorial"; <http://java.sun.com/developer/onlineTraining/java3d/>
- [J3DSpecification] Sun Microsystems, "Java 3d API Specification"; <http://java.sun.com/products/java-media/3D/releases.html>
- [The joy of Java3d] Greg Hopkins, "The Joy of Java 3D"; 2001, <http://www.java3d.org/introduction.html>