Henry van der Beek
Crowd Scene
Report

1. Contents

1. Contents 2
2. Introduction 3
3. Background 4
   3.1 Concept 4
   3.2 Story 5
   3.3 Bullring Architecture 5
4. Research 7
   4.1 Concatenating Animation 7
   4.2 Crowd Simulation 7
      4.2.1 Craig Reynolds (Flocking) 7
      4.2.2 Softimage Behaviour 8
      4.2.3 Massive 8
5. Pipeline 9
   5.1 Animation 9
   5.2 Modelling 10
   5.2 Modelling 10
      5.2.1 Bullring 10
      5.2.2 Other Buildings 10
   5.3 Simulation 11
      5.3.1 Outline 11
      5.3.2 Class Design 11
      5.3.3 Process 15
   5.4 Rendering 20
      5.4.1 Generate mtor ribs 20
      5.4.2 Append biped ribs 20
      5.4.3 Render 21
      5.5 Compositing 22
6. Further Work 23
   6.1 Uneven Terrain 23
   6.2 More Animation Cycles 23
   6.3 Simulate on Skeleton Level 23
   6.4 Fuzzy Logic 24
7. Bibloigraphy 25
   7.1 Academic Papers 25
   7.2 Online Resources 25
   7.3 Software Packages 25

APPENDIX: Storyboards 26

2. Introduction

The purpose of this project was to create an animation using a crowd scene simulation. Due to the volume of processing required in scenes containing large amounts of geometry, this was to be done using distributed rendering.

3. Background

3.1 Concept

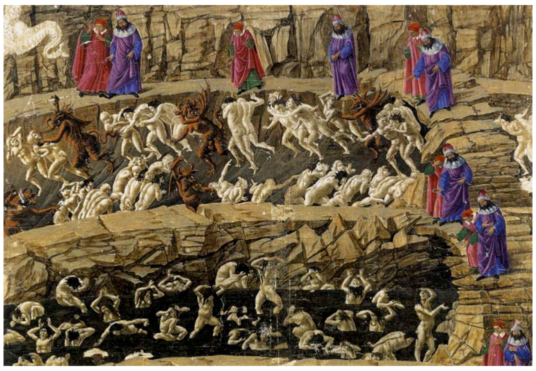
The initial idea for this project was inspired by a woodprint called "Malaga Bullring" by Emma Stibbon (Picture 3), which was in an exhibition on the ground floor of Poole House in Bournemouth University. I like the harsh lighting and the simple majesty of the print, and this is what made me decide to base my project on it.

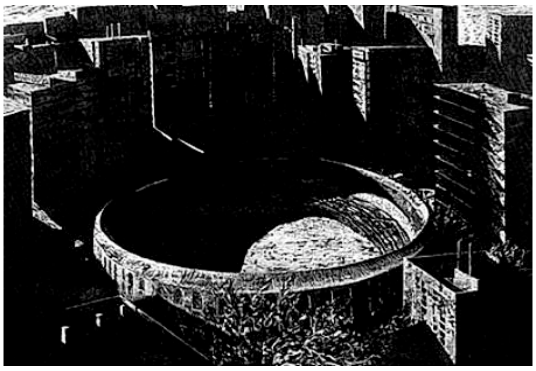
This was to be used as the basis for a crowd simulation, and I decided that I wanted to create an animation of a crowd entering the bullring. This reminded me of an experience I had whilst travelling in Brazil. I was on a bus travelling through Agua Fria, a poor suburb of Recife in the North East. There was a football match on, and the hot air was full of dust and smoke, and people shouting and screaming. The stadium loomed out of the fog, made from raw grey concrete and covered in graffiti.

I was struck how the crowd seemed to move as one faceless mass. The energy and aggression washed over everyone as a wave, and drew them onwards into the stadium. It reminded me of Bosch's paintings of hell (Picture 1) and Botticelli's visions of Dante's inferno (Picture 2).

This was the kind of crowd that I was aiming to simulate.

Picture 1: The Garden of Eartly Delights (Panel 3) (Hieronymous Bosch c. 1500)


Picture 2: Canto XVII (Sandro Botticelli c. 1480)


Picture 3: Malaga Bullring (Emma Stibbon 2002) woodcut print, 112 x 153 cm

3.2 Story

From these initial ideas I developed a simple story to base my simulation around.

The story starts with a crowd flowing into the bullring. When the crowd were all seated, it is revealed that a monster is lying sleeping in the centre of the auditorium. The crowd jeer and taunt the monster, who paces around uncomfortably. Then, the monster finds that he can reach up to the edge of the seats and climb up. As crowd begins to flee in terror, the monster grabs one of them and eats it. The scene ends, with the monster running through the panicking crowd, eating people (see Storyboards Appendix).
.

The music chosen for the scene was “Rusalka’s Ode To The Moon”, from the opera “Rusalka” (1900) by Antonín Dvorak. In this, Rusalka sings to the moon first in awe, then in reassurance. This represents the monster’s transition from melancholy to content.

Silver Moon upon the deep dark sky,
Through the vast night pierce your rays.
This sleeping world you wander by,
smiling on men’s homes and ways.

3.3 Bullring Architecture

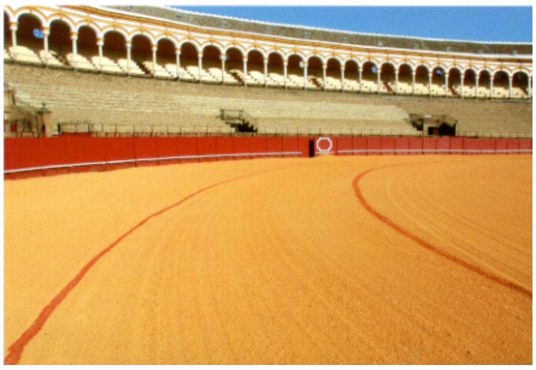
The first stage in modelling the scene was to look into the designs of the Malaga Bullring and its surrounding area. From google maps, I was able to get a satellite picture and a map.

During the course of the project I also took a trip to Seville, the town in Spain most famed for bullfighting. I took the opportunity to visit the bullring, and took several photographs. I also managed to get a picture of a schematic of the seating stalls, which was useful in the modelling (Picture 11).

Picture 4: Malaga Bullring aerial view (source: google maps)

MALAGA > Callejero de MALAGA

Picture 5: Map of area surrounding Malaga Bullring (source: google maps)

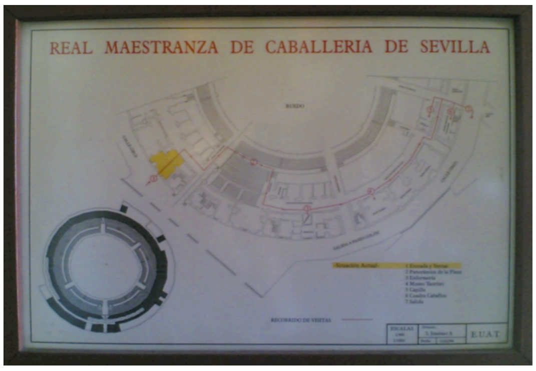


Picture 6: Seville Bullring interior (postcard)

Picture 7: Seville Bullring interior (postcard)

Picture 8: Seville Bullring tunnels (photograph)

Picture 9: Seville Bullring exterior (photograph)



Picture 10: Seville Bullring schematic (photograph)

Picture 11: Seville Bullring schematic (photograph)

4. Research

4.1 Concatenating Animation

A built in feature of Maya, the Trax editor, allows animation to be transferred between characters, and to be duplicated across the same character, for example if you wanted to repeat a walk cycle a number of time.

However, I found the Trax editor very problematic and not very well documented, so I wrote a simple script for exporting animation files in my own format (.clip files). This worked by going through a skeleton hierarchy for a character, and finding either the position or each keyframe for that particular joint. Another script was then used to write the .clip files to new bipeds, thus duplicating the animation.

Unfortunately when this was applied numerous skinned characters, maya seemed to find it hard to handle all of the information, even if the viewport was not updated during processing, and undo was turned off. Scenes with more that 100 bipeds became unmanageable and it was decided to switch to Renderman to place the biped skins in the scene. This rules out the possibility of controlling the bipeds on a skeletal level.

4.2 Crowd Simulation

4.2.1 Craig Reynolds (Flocking)

Craig Reynolds is famous for his work in both the fields of animation and artificial life. In 1987 his paper “Flocks, Herds and Schools: A Distributed Behaviour Model” was published. The paper applied simple rules to the describe behaviour birdlike creatures called boids. This resulted in seemingly complex behaviour of the flock, produced by the simple behaviour of the individual.

Reynolds' idea stemmed from an interest in the concept of autonomous characters. While examining the behaviour of a flock of birds, he imagined himself flying as a member of the flock. Assuming that as individuals, the birds were not especially intelligent, Reynolds came up with three simple rules which would allow them to continue flying together as a flock. Firstly, they would want to avoid colliding with other birds. Secondly, they would want to keep moving at the same pace as the rest of the birds, and finally, if they were drifting away from the flock, they would alter their direction towards the rest of the flock. These three rules: Collision Avoidance, Velocity Matching and Flock Centering; formed the basis of Reynolds' work. In applying these, he found that he could quite accurately model the behaviour of a flock of birds.

The paper then went on to outline how he had experimented with this behaviour. Independent of any other criteria to cause behaviour, he found that the flock tended to reach a steady or state in which no new behaviour was exhibited. To liven up the flock, Reynolds introduced a “global direction” to model the flock's will to fly towards a particular place, for example to migrate during the winter. This took the form of a virtual carrot on a string, or a tendency to fly towards a set “goal point” which would move along a particular path around the scene. Later in the paper, another feature was introduced: obstacles for the boids to avoid. The goal point was then used to steer the boids towards these objects, causing the flock to split, and then reform, once the obstacle had been circumnavigated.

4.2.2 Softimage Behaviour

Behaviour is a crowd simulation package designed to be used with XSI. It incorporates character terrain following, obstable avoidance and ragdoll stuntmen.

The characters and actions are modelled in XSI and then loaded into a behaviour scene, where the final animation is constructed. Behaviour objects can be one of three types of “actor”, that is any object which interacts in the scene. These are:

- Simple Props: inanimate objects to be avoided.
- Terrain: objects which movement curves must follow (only one terrain per scene).
- Digital Actor: characters which move in the scene, as controlled by Behaviour.

Each Digital Actor’s behaviour is defined by a finite state machine, which is constructed as a series of linked nodes. Using this, it is possible to make an actor behave differently depending on their proximity to an enemy, for example.

4.2.3 Massive

Massive (Multiple Agent Simulation System in Virtual Environment) is a piece of crowd simulation software which was written by initially written by Stephen Regelous for use in Lord of the Rings. It now exists as a standalone application and has been used in many films and advertisements.

Similarly to Behaviour, character AI is build up in a series of nodes. However, in Massive, the nodes are broken individual behaviour nodes (or brain), and group behaviour nodes, dictating the personality of the group as a whole. In addition to this, the system is on Fuzzy Logic, rather than a finite state machine, and so outputs a much wider and more realistic array of behaviour.

5. Pipeline

5.1 Animation

The whole basis of the final piece is the copying of animation to many places within the scene, managed by the simulation. This animation was done in maya using a simple skeleton with a smoothly bound polygonal skin, with some weights painted onto to body to get satisfactory deformations.

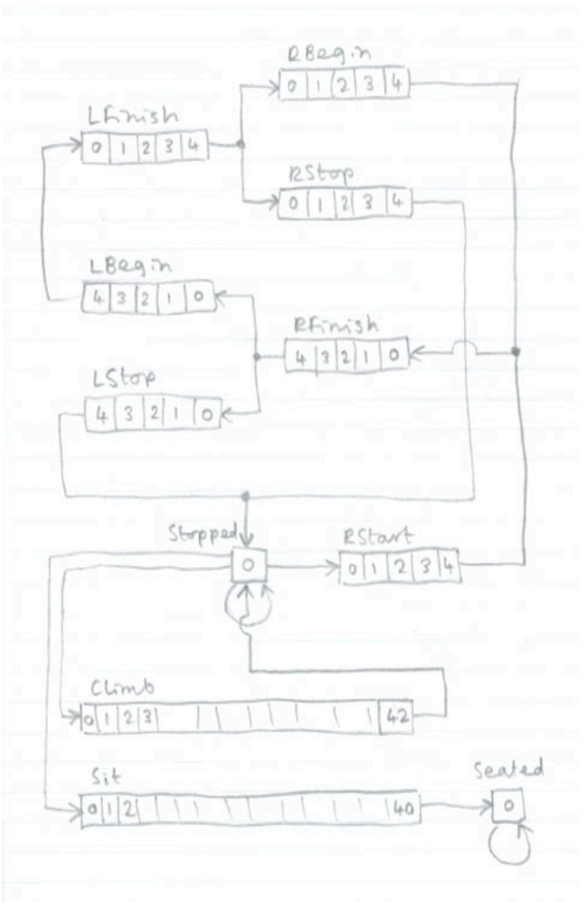


Diagram 1: Animation cycles. Bipeds always start from Stopped, and begin walking on their right foot (RStart), which then leads into the closing of the step (RFinish). From this position, they can either move into a left step (LBegin) or into a stopped position (LStop -> Stopped), etc.

Several key poses were chosen (stopped, left foot forward, right foot forward and seated), and these poses were used as start and end points for each animated sequence, to allow the blending of different pieces of animation.

For all of the walking and stopping animation, the main axis of the biped was kept in the same place, so that any movement could be controlled by the simulation. However, in the cases of the longer animated sequences (the sitting and the climbing), the biped was allowed to move relative to the origin, so that the simulation could maintain a constant position and then apply a step increment when the sequence was complete.

Each sequence was then rendered out as a series of rib files using the mtor rib export feature.

5.2 Modelling

5.2.1 Bullring

As the bullring was to be defined in the simulation as a series of geometric shapes (circles and lines) for the simulation, it was necessary to model the bullring mathematically to make sure that the model and the simulation data matched up.

To allow this, the whole bullring structure was generated by a mel script (**generate_bullring.mel**). This script creates the shape bullring based on the values of nine variables, which are set in the text of the script. These are:

- Outside Radius (outer_rad)
- Inside Radius (inner_rad)
- Central Passage Radius (centre_rad)
- Centre Passage Width (cpass_w)
- Outer Tunnel Width (rpass_w)
- Inner Tunnel Width (spass_w)
- Passage Height (pass_h)
- Number of Entrances (num_ents)
- Number of Seating Rows (num_srows)

The height of the seating rows was not defined in the script, as this was already defined as 2.0 units by the animation. A button was made in the maya shelf so that the bullring could easily be generated after minor adjustments were made to the shape in the script.

The script works by generating cubes and cylinders of various sizes, and then using Boolean geometry operations to combine the pieces into the required bullring shape.

As well as creating the model, the script is also responsible for generating **bullring_spec.txt**, a text file containing the values of these nine variables, to be read in by the simulation.

5.2.2 Other Buildings

Other buildings were modelled roughly based on the Emma Stibbon woodprint, and photographs of the Malaga Bullring taken from the hills to the north. I felt that accurately modelling the buildings was not necessary, in keeping with the harsh, simplistic tones of the woodprint

Once the buildings were modelled, key vertices (ie. the corners) were chosen as reference points for importing the buildings into the simulation. Another mel script, **write_out_locators.mel**, was used to write out the global positions of these points to a text file (**scene_locators.txt**), which was read in by the simulation.

Once again, a maya shelf button was made to output the locators, allowing the buildings to be moved and scaled in the scene and keep the simulation up to date with their positions.

5.3 Simulation

The **Bullring** executable formed the basis of the project. It was written in C++, and can be adjusted at the command line for different numbers of bipeds, placing of bipeds and numbers of frames. The simulation is then output in the form of two types of text files, one for maya to read (**Sim.txt**), and the other to be rendered by Renderman (**biped.####.rib**).

First I will outline the overall design of the simulation, and then I will go into more detail about the classes used. Then I will explain the process of the execution of the program.

5.3.1 Outline

The simulation models the movement of a number of bipeds as they walk through an environment containing several objects. Using the same notation as XSI Behaviour, every interacting object in the scene was called an actor; in the program, all of these classes inherited from the base Actor class.

In general, the collision testing is done in two dimensions, thus all positions are stored as Vector2 objects. When the bipeds enter the seating, some 3 dimensional collision testing is applied, but still the x and z horizontal axes are kept separate from the vertical y axis.

5.3.2 Class Design

Overview

The scene consists of the members of the crowd, which are called Bipeds, and a number of Areas, which are defined as “areas in which the Biped can exist”. A Biped’s movement depends primarily on which area it is in. For example, if a Biped is in the Outside Area, it will walk towards the closest entrance to the bullring. In addition to this, Bipeds also avoid each other and collision objects within their Area.

Class Diagram

As the class structure is rather complicated, I have broken down the class diagram into separate inheritance and usage diagrams.

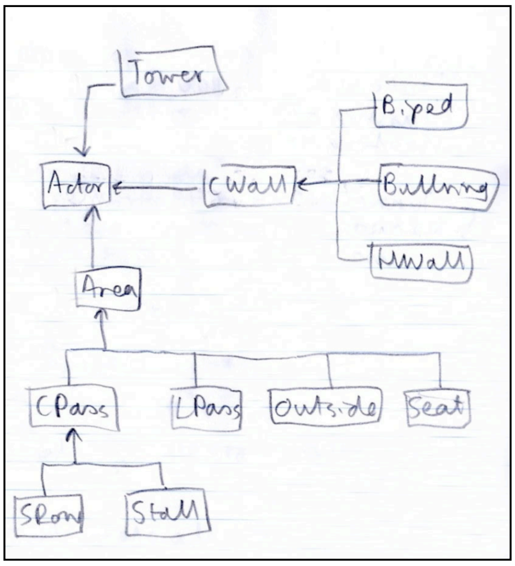


Diagram 2: Inheritance Diagram

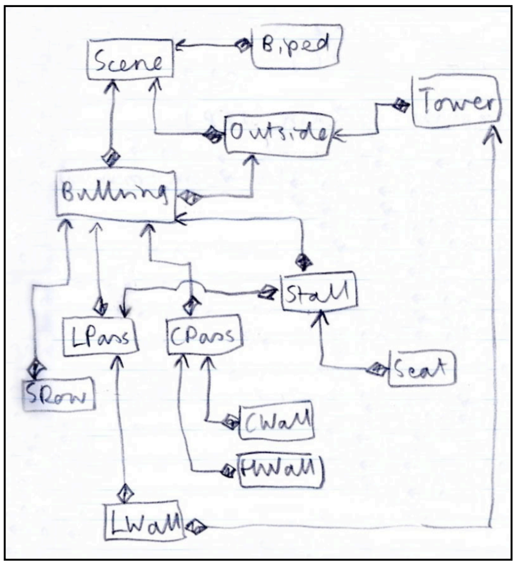


Diagram 3: Usage Diagram

Actor

An actor is defined as any object which is able to interact in the scene, from a biped to the bullring. This is a generic abstract class which all elements in the scene inherit from.

Though the simulation is primarily kept in two dimensions to minimise the processing, the each Actor class contains a several vectors of three dimensional objects (vertices, edges and polygons) to allow the scene to be visualised. These objects are not used in the collision tests.

CWall

A Circular Wall is a circular object in the scene used for creating collision boundaries, defined by a position and a radius. Objects which are within the CWall's radius are considered to be colliding, and objects outside the radius are considered not colliding.

HWall

A Hole Wall is the opposite shape to a Circular Wall. Objects falling outside its radius are considered collided, while objects within the radius are not.

LWall

A Linear Wall is defines a plane across the scene. It is defined by a position and a normal. An object on the side of the plane behind the normal is considered to be colliding.

Biped

Each Biped object represents a character in the crowd. The biped inherits from the CWall class, giving it a circular collision area. Each object maintains information about it current location and position in the animation sequences. This information is then updated each frame.

Area

An Area is defined as a location which a biped could inhabit or avoid, eg. a building or a passageway. This itself is a type of actor, so inherits from the Actor class.

The abstract Area class has several important virtual methods.

With regards to collision testing, `Area::intesects` (whether the object is partially within the area) and `Area::contains` (if the object is contained within the area) are the key methods. These tests can be performed in either two or three dimensions.

The bipeds' area also defines their movement. There are several methods relating to this such as `Area::get_direction`, `Area::get_destination` and `Area::get_next_area`.

Another important role of the Area class is the spawning of Bipeds at scene initiation. The two methods controlling this are `Area::get_random_position` and `Area::randomly_place_biped`.

CPass

A Circular Passage contains two walls, an outer HWall, and an inner CWall. Using these two boundaries, it can determine whether an object lies within it.

LPass

Like a CPass, Linear Passage has two walls. However, in this case, the two walls are LWalls facing each other. An object is inside the LPass if it is inside neither wall.

SRows

The Seating Rows are types of CPass with special properties. In other areas, Bipeds can enter the next area simply by walking, as the areas overlap. However, the SRows are a series of concentric CPasses, with every outer wall the inner wall of the next row. Because of this, the `Area::allow_climbing` method returns negative for all Areas apart from SRows. This allows the bipeds to check whether it is possible to climb onto the next row up, but only in these areas.

SRows are also responsible maintaining a list of seats which have been allocated to Biped within their boundaries.

Seat

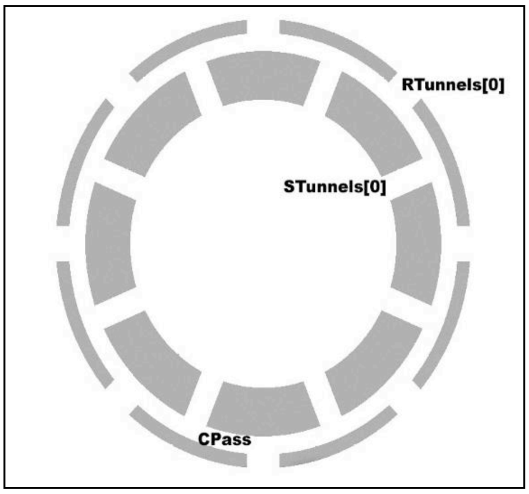
A Seat object is a type of CWall. It exists as a target for the Biped to whom it has been allocated, and a collision object for Bipeds requesting new seats. It has a radius slightly larger than the biped requesting it, to allow the biped to step inside it.

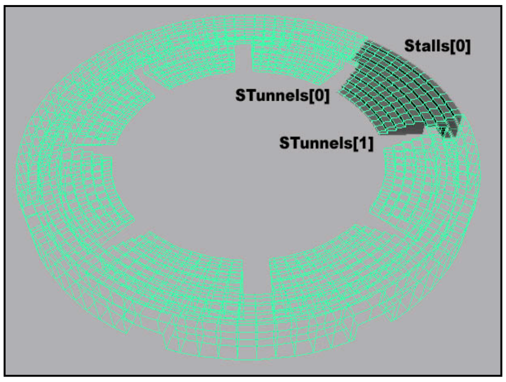
Tower

A Tower is a collision object representing a tower block building. It consists of four or six LWalls. An object is only considered as colliding with the tower if it is inside all of its walls.

Stall

A Stall is a type of area used in seating management. It inherits from CPass, but it also has a sweep value, limiting its range. The bullring seating consists of several stalls, and the sum of their sweeps in 360 degrees.





(left) **Diagram 4:** Passages inside the bullring
(above) **Diagram 5:** Stalls in the bullring

Bullring

The Bullring object represents all of the seats, walls and internal passageways of the bullring. It has a circular hull, and a number of entrances even distributed around its exterior. Each entrance is represented by an LPass called `RTunnels[n]` for historical reasons. These LPasses all lead into one internal CPass called `CTunnel`. This then has several more LPasses leading from it, called `STunnels` (see diagram 4). There are the same number of `STunnels` as `RTunnels`. All of the `STunnels` lead into the first of the SRows, `SRows[0]`. The SRows are then numbered sequentially upwards and outwards. Each STunnel also has two Stalls, one to the left and one to the right (see Diagram 5). Each stall covers all of the SRows between each STunnel.

Outside

The Outside object contains all objects affecting the biped when they are outside of the bullring. This consists of four towers and the hull of the bullring, which are all applied as collision geometry.

5.3.3 Process

Initiation

First a scene object is set up, and then the `Scene::Init` method is run. This performs the following tasks:

Environment Generation

First, the bullring area is created by creating a Bullring object. This reads in the **bullring_spec.txt** file, which contains the up-to-date specifications for the shape of the bullring. From this data, the various walls, passages, stalls and seating rows are created.

Next, an Outside object is created to represent the outside area. The positions of the towers are read in from the **scene_locators.txt** file, and once again, the necessary walls are constructed to place these in the scene. A reference to the bullring object is also passed to the outside object, as the bullring is a collision object in the outside area,

Biped Generation

Bipeds can be generated in any of the areas. The number generated is set by the –b flag on the command line, the default being one. The required biped density in each area can also be set at the command line, using the –d flag (default is 0.1). This is only used as a limiting value though; the density in any area will be limited first by the total number of bipeds in the scene and no area will be forced to reach a specific density.

A stack of Areas is created as the spawn areas. This can be adjusted for different shots, but only at a coding level. For example, for an exterior shot, it may only be necessary to spawn bipeds in the outside area.

Spawning is performed using the `Scene::fill_area_to_density` function. When an area's density reaches the threshold value, the area is pushed off the stack, and spawning continues in the next area. The spawning function also takes in a `sweep_min` and `sweep_max` values, to allow bipeds to only be generated in certain areas. This is useful for interior shots, where it is not necessary to completely fill the bullring as the camera can only see a small part of it.

The `Scene::fill_area_to_density` function also has the functionality to change the distribution of the spawning of the bipeds, and supports linear, quadratic, cubic and quartic distribution. This feature is useful in spawning bipeds in the outside area; using a non-linear distribution will cause a denser population around the bullring, and a sparser population further away. This looks more natural and avoids a sharp line of bipeds at the edge of the spawning area.

When a biped is spawned, it is tested by the scene to see whether it collides with any of the other bipeds. If this test fails more than 1000 times, the scene assumes the area is full and rejects it, pushing it off the stack of spawn areas.

Bipeds are generated as stopped to begin with, and are told to wait for a random number of frames in the range of the length of the walk cycle. This is so that their steps are not all synchronised.

When bipeds are generated in the seating rows, this is a special case. To prevent intensive demands on the number of seats (see later), the bipeds are generated as already seated. This requires them to be generated in specific positions on the seating rows, and this allows only a smaller number of bipeds to be generated. For this reason, the biped densities in the seating rows are scaled down for spawning.

Runtime

Update

Destination

Each biped's movement is defined by the area it is in. Each biped has an attribute `currentArea`, which it asks for its `destination`. For example, if the biped asks the outside area for its `destination`, the outside area will return the nearest bullring entrance to that biped. Likewise if the biped is in the seating row containing its seat, the seating row area will return the position of its seat as its `destination`. The `direction` is then given as the vector between the biped and its `destination`, normalised.

As well as storing its `currentArea`, each biped also has an attribute `nextArea`. If the biped has entered its `nextArea`, the `Biped::process_area_transition` method is called, which makes the `currentArea` the `nextArea`, and requests this area for the subsequent area using the `Area::get_next_area` method.

Once again, when the biped enters the seating area, the situation is slightly different. Their `currentArea` is the seating row which they are currently in, but their `nextArea` checks the position of their seat. If it is in the same row that the biped currently occupies, the `nextArea` is set to the biped's seat. Otherwise, the `nextArea` is set to the next seating row up.

Steps

To make the calculations less intensive, bipeds only perform a collision test when they are about to take a new step, which is every 12 frames. At this point, they propose a new step towards their direction, and send a request to the scene to allow this step. This uses the `Scene::allow_step` method. The scene then tests the new step against the biped's area's collision geometry and not only all the other bipeds in the scene, but also the position of their next steps. This was to simulate how people in a crowd are able to judge where other people are going, and avoid stepping into their path.

If the next step is rejected, the biped traverses a list of other possible steps which it can make. These are steps of varying angle and distance at 20 degree and quarter step increments. These are sorted in order of distance towards the destination and are part of the biped object, generated at runtime into two steps vectors. The biped will chose the first possible step which it comes to, this being the one which will take it closest to it destination. If all of the steps are rejected, the biped will stop.

The steps vectors are split into two so that the biped will not readily take steps which are at oblique angles to its destination. All of the steps taking the biped a set distance towards its destination are stored in the `possible_steps` vector. These steps are available every time a step is requested. All remaining steps, taking the biped less than the threshold towards its destination, are stored in the `drastic_steps` vector. These are only made available when the biped has been stopped for more than 10 frames. This system prevents the bipeds from moving around erratically if there is something in their path.

Animation

Each biped's animation is dictated by the simulation, and is output in the form of renderman code, with an attribute box for each biped, and a separate file for each frame. Each biped's code will look something like this:

```
AttributeBegin
      Translate x ypos z
      Rotate angle 0 1 0
      Surface "rmanshader/Blinn_0"
      Procedural "DelayedReadArchive"
            [animation_file] [-10 10 -10 10 -10 10]
AttributeEnd
```

The DelayedReadArchive function references a rib file which was generated for each possible biped pose (see the Section 5.1). The simulation applies an animation file depending on the biped's `animation_state` attribute and their position within that state as defined by their `sequence_counter` attribute. For example, if a biped is in a **CLIMBING** state, and the `sequence_counter` is on 38, we know that the animation file to be referenced is Climb.0002.rib, since the Climb animation is 40 frames long.

At each update, the `sequence_counter` is decremented. When it reaches zero, the biped requests a new state using the `Biped::process_animation_transition` method. This then assigns a new animation state, and sets the `sequence_counter` to the value of the duration of that state.

The state transitions roughly follow the animation sequence diagram (see Diagram 1), and in most cases, the animation frame is written on the same frame as it is simulated.

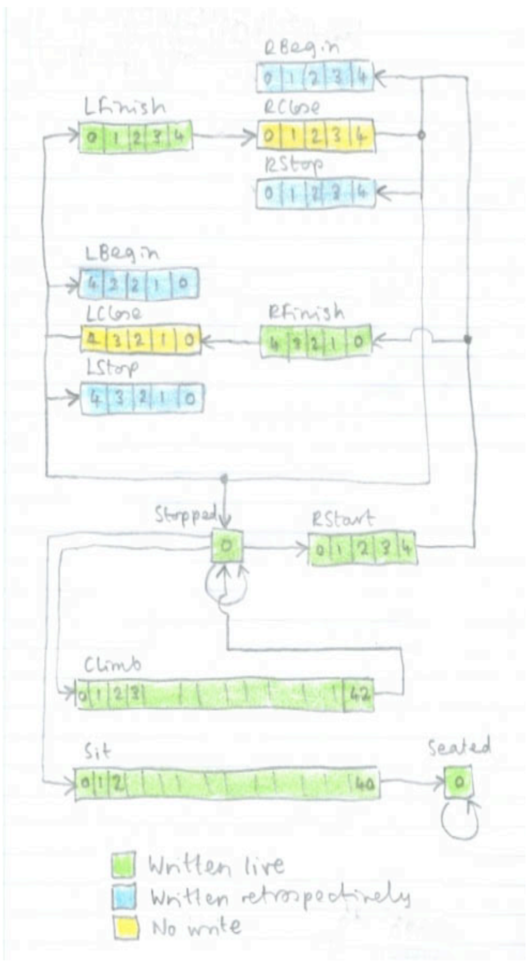


Diagram 6: Animation state transition diagram to allow retrospective writing of frames.

However, when a biped moving its feet together is going to have its next step accepted or rejected, and in this case, we do not know whether to apply the "Stop" or "Begin" sequences. This means that our animation sequence diagram needs some adjustment for writing out the animation (see Diagram 6).

In essence, a virtual animation state is created called "Close". During this state, no animation is written, but at the end the five frames leading up it are written all at once. Though this makes the bipeds seem to react to things before they have happened, it is an effective approximation of a person's judgement of the people walking in their vicinity.

Due to this retrospective writing of frames, the simulation needs to have 6 file streams open at any one time, one for each of the pervious five frames, and one for the current frame. This is managed by the scene object. The bipeds then write themselves to the rib files for each frame using the `Scene::write_biped_to_file` method.

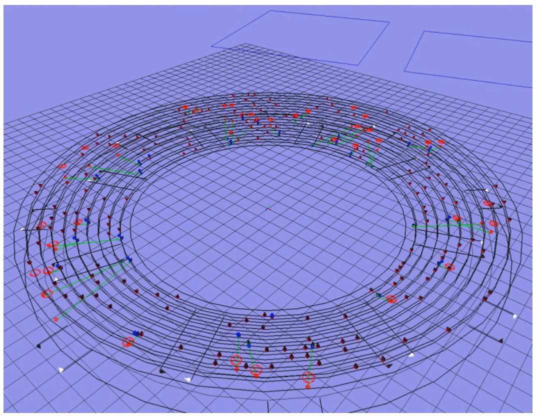
Seat Allocation

Seats are allocated to an actor when it enters the auditorium. Each passage leading into the auditorium has two Stall attributes, `left_stall` and `right_stall`. If the biped is on the left side of the passage entering the auditorium, it is allocated a random seat in the left stall, and likewise for the other side.

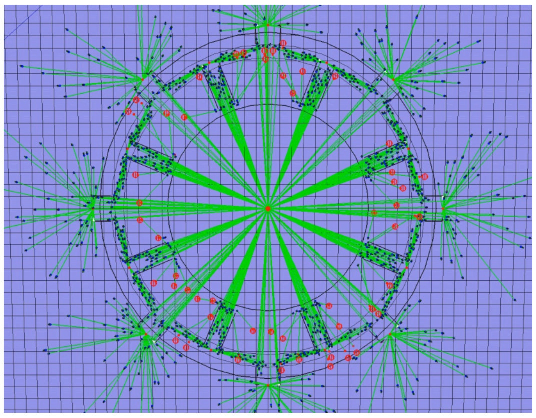
Sometimes the bipeds get stuck in a position trying to get to their seat, particularly if they have just entered the auditorium and there is someone sitting right next to the entrance. To prevent this, if biped are stopped for more than 50 frames, they are allocated a new seat each frame until they start moving.

Draw

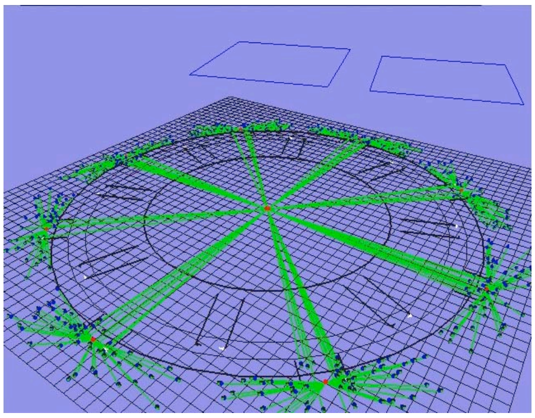
Each actor in the scene inherits the `Actor::Draw` method. This goes through each of the object's edges, faces and vertices and draws each one. For visualisation of the simulation, it was found that using the edges was most effective so that the user could see inside the bullring.



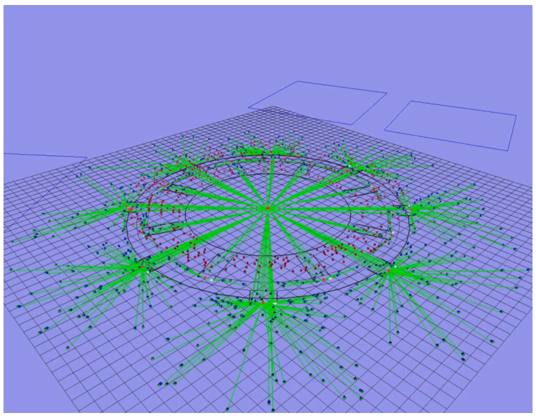
Picture 12: Simulation with most of the bipeds seated. The black dots are seated bipeds, the blue dots are bipeds which are climbing. The red areas are seats which have not yet been reached.



Picture 13: An overhead view of the simulation. Green lines show the line between the biped and its destination. Notice bipeds outside the bullring are all moving towards their closest entrance.



Picture 14: The square grid shows the radius in which the bipeds are generated. In the background you can see to collision geometry of the towers.



Picture 15: In this simulation, bipeds are evenly distributed across all areas.

Write

The write part of the update cycle generates a text file called **Sim.txt**. This is a version of the simulation which can be read in by maya. To minimise the amount of reading, only bipeds changing direction or stopping/starting were written to the file. This was achieved by having a `scene.bipeds_to_key` vector, and only pushing bipeds which need to be keyframed onto it. At the end of each update, these bipeds are written out, and the vector is emptied.

5.4 Rendering

5.4.1 Generate mtor ribs

Read in simulation

This was an optional part of the pipeline. Using the **readSim_to_locators.mel** script, the positions of each biped are imported into the scene and applied to locators. However, with large crowds, mel's parsing of text files and writing data to the scene was so slow that it was often ineffective to use this feature of the project.

Passes

Render passes were set up in slim. The shots were done with two passes.

The first was a crowd beauty pass. All objects in the scene were shaded with the Matte attribute turned on. This was applied through a RIBBox shader. The scene lights were left one.

The second pass was an ambient occlusion pass. A shader was imported into slim, and applied to all surfaces in the scene. For this pass, all of the lights were turned off.

Mtor Render

Camera moves were applied, and the frames were then rendered out using mtor.

5.4.2 Append biped ribs

Joining ribs

In some cases in was necessary to join the ribs from two different simulations into one shot. For example, in the seating shot, the bipeds in the auditorium and outside were simulated in different batches. This was done using a simple python script called **join_ribs**.

Create Final ribs

The final fibs are constructed by inserting the biped rib files into the mtor ribs. The function was performed by the **Crowd_Scene_Render** python script, using the **bullring_FM.py** (bullring file manager) module.

On activation, the **bullring_FM** module tests whether the biped files in has in its buffer directory are up to date by comparing the md5sum value of its files and the files in the simulation output directory. If the files are not up to date, it queries the user to determine whether an update is required. If so, it copies the files across to its buffer.

The **bullring_FM** module also allows an offset to be applied to the biped rib files. This is so that biped files can be used from a simulation that has been running for a few hundred frames can be used, so that the positions seem more natural. This is activated by using the –bo flag in the **Crowd_Scene_Render** script.

Next, the **bullring_FM** module creates a `.Final_Ribs/` directory in the project area, and begins to generate the final ribs into this location. This process uses simple `head`, `grep` and append (`>>`) command line options.

Upon generation, each biped rib is submitted for rendering, which is explained in the next section. This is so that the whole list of ribs need not be traversed before rendering can begin.

5.4.3 Render

Find free machines

If **Crowd_Scene_Render** is performing a distributed render, it activates the **ninfarm.py** module, which constructs a list of available machines. This can be done in two ways.

The default is to read the machine list from a file, stored in the **ninfarm** home area. This is appropriate when the list has been recently updated, and when not many people are moving in and out of the labs.

If the **ninfarm** is instructed to refresh the list, it scans through a list of all available machines on the ground floor of Weymouth House. By default, it rejects machines if people are logged in, but if it is run in `force` mode, it checks the CPU usage of machines which people are logged in to, and allows the operator to force these machines onto the list. This is appropriate if someone's rendering has finished for example.

Submit jobs

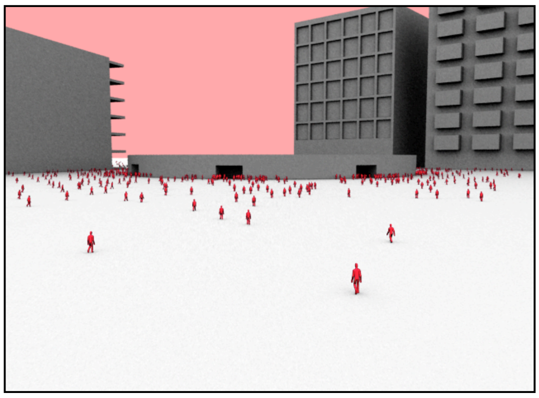
Each render job is submitted to the **ninfarm** using the `ninfarm.get_next_machine` method. This uses the **ninwho.py** module, which is used to check usage statistic of a machine.

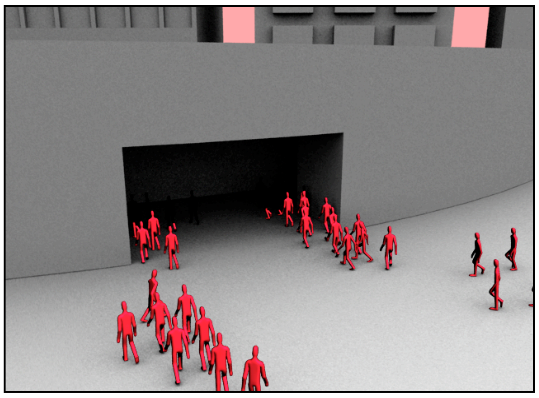
The functionality of the **ninwho** module revolves around minimising the number of ssh requests to a machine, saving time. Every time **ninwho** requests the usage of a machine, it stores the outcome in its home area, along with the time of the usage request. The module then offers features to extract different statistics from this cached data.

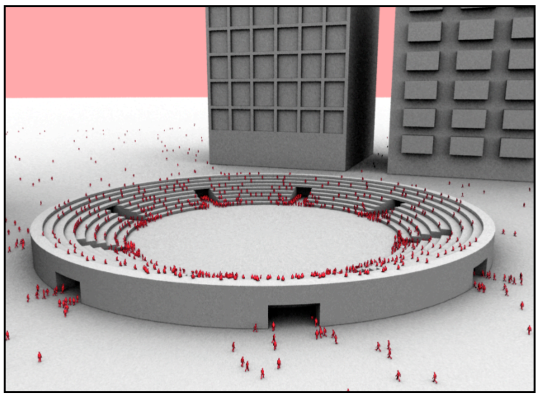
Using the **ninwho** method, the **ninfarm** submits frames sequentially to available machines. A frame is considered rendered when all instances of prman processes have completed on that machine.

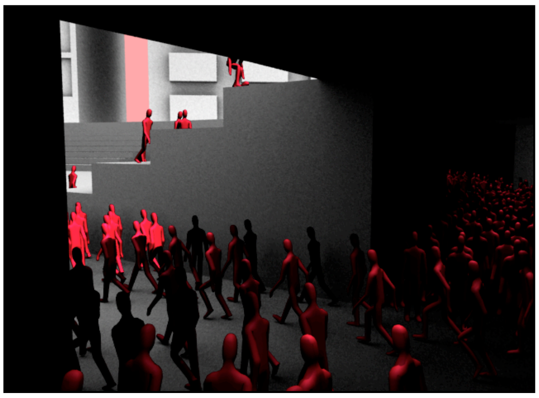
5.5 Compositing

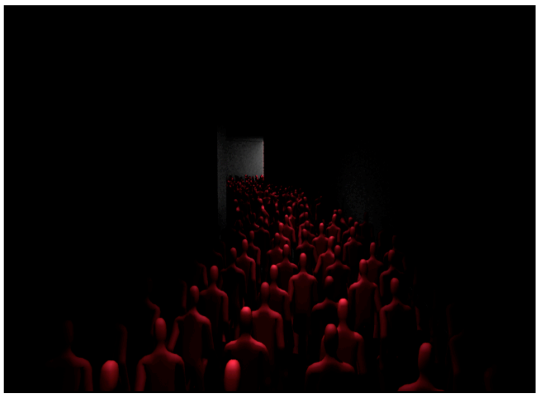
Compositing work was done in Shake. The two passes were joined together using an over node, and the background sky colour was put in. The frames were then rendered out as the finished piece.

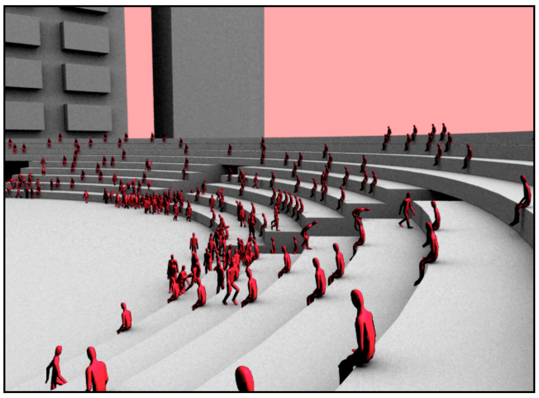

Picture 16: Long shot of the scene.


Picture 17: Bipeds entering the bullring entrance.


Picture 18: Bullring full of people.


Picture 19: Bipeds entering the auditorium.


Picture 20: Bipeds in the circular tunnel.


Picture 21: Close up of the seating.

6. Further Work

6.1 Uneven Terrain

At the moment, my scene looks rather flat. To introduce a small level of unevenness into the terrain would be quite easy, since you could simply use the current animation cycles and tilt them to the normal of the terrain surface at that point.

To introduce more rugged terrain would also be nice, but would require the simulation to be run on a biped skeleton level (see below) as large increments in terrain heights would require different deformations in the biped skin.

6.2 More Animation Cycles

To allow some diversity in the crowd, multiple animation cycles could be made. The simulation would then randomly choose a character for each member of the crowd, and apply those animations to that character.

This would be quite work intensive, but would easily fit into the current framework and produce some nice results.

6.3 Simulate on Skeleton Level

When I first began working on this project, I entertained thoughts of running the simulation on the level of controlling biped's leg movements, this in turn controlling their arm movements, and thus defining the movement of the whole skeleton. However, the when the number of people in the scene became too much for maya to handle, I lost the ability to bind the skin to the skeleton, save running each character through maya, which would be very time consuming.

One way around this would be to use a script to generate several thousand rib files for each possible postion. For example, if you wanted to simulation a turn, you could simply generate the ribs for the sequence for each one degree increment, resulting in 4320 ribs for a cycle of 12 frames, which would be quite practical as they are quite small. The simulation would then pick the ribs appropriate to the turn being applied. One could even experiment as to the maximum degree of increment which was noticeable.

Another solution would be to bind the skin in the simulation. This would probably be more efficient, but would require (for me personally) a whole new area to research into the practical and mathematical aspects of binding a skin to a skeleton.

6.4 Fuzzy Logic

One of the key strengths of Massive is the use of fuzzy logic.

While my simulation is based on a finite state machine, massive uses fuzzy logic, allowing an actor to exist in any number of states to varying degrees, at any given time. This in turn creates an infinite number of possible animations which a given character can perform.

Once again this would require the simulation to be performed on skeleton level, but would open up a whole new world of possibilities and realism.

7. Bibloigraphy

7.1 *Academic Papers*

● Reynolds, Craig W. Flocks, Herds, and Schools: A Distributed Behavioral Model 1 Computer Graphics, 21(4), July 1987, pp. 2534.

7.2 *Online Resources*

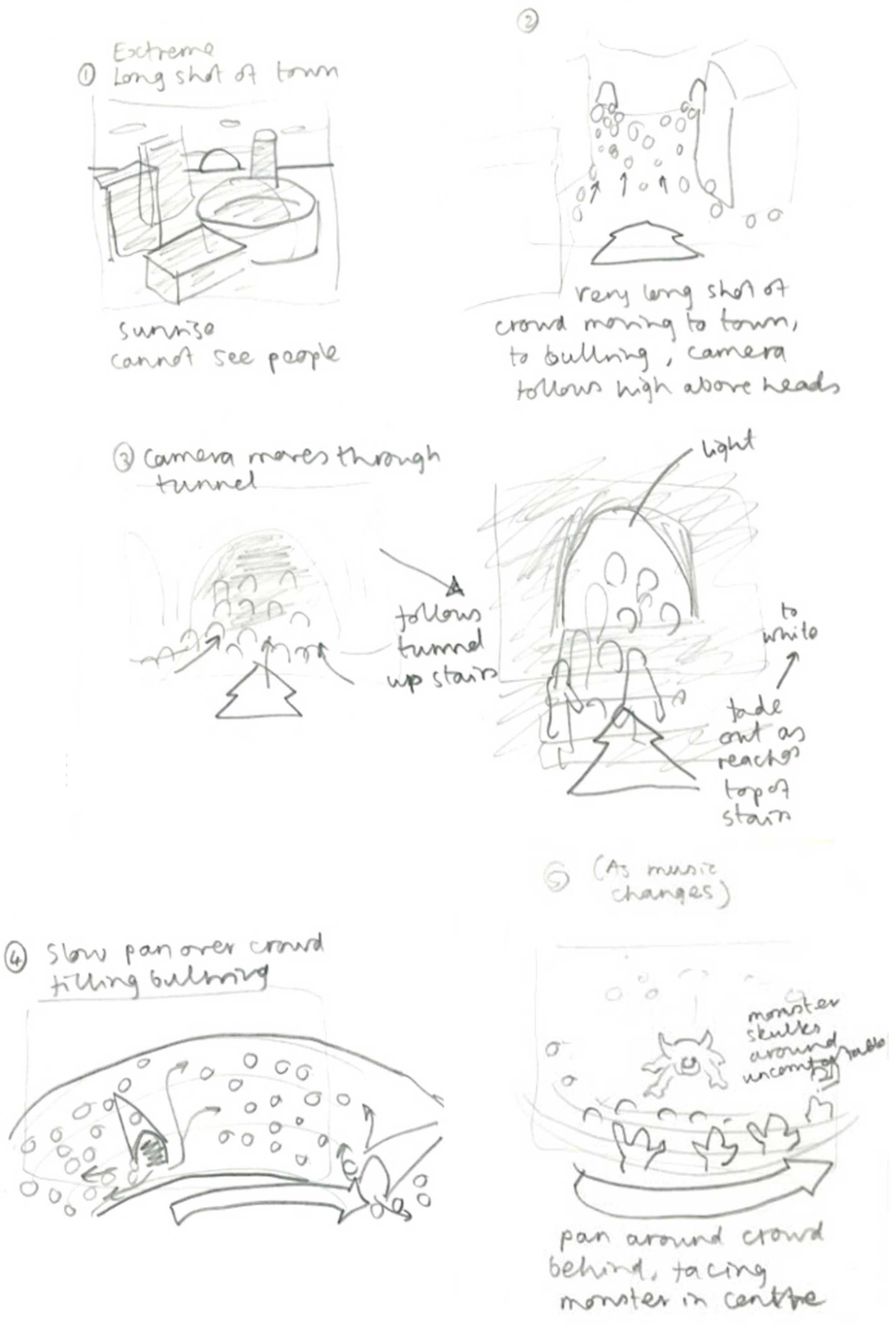
[Encyclopedia] http://en.wikipedia.org
[Article on Craig Reynolds] www.generation5.org/content/1999/reynolds.asp
[Various articles about Renderman] http://www.fundza.com/

7.3 *Software Packages*

Maya 7.0 (Alias)
c++ (Free distribution)
Renderman (Pixar)

XSI (Softimage)
Behaviour (Softimage)

APPENDIX: Storyboard


① Extreme Long shot of town
sunrise cannot see people
②
very long shot of crowd moving to ~~town~~, to bullring, camera follows high above heads
③ camera moves through tunnel
follows tunnel up stairs
light
to white
fade out as reaches top of stairs
④ Slow pan over crowd filling bullring
⑤ (As music changes)
monster skulks around uncomfortable
pan around crowd behind, facing monster in centre

⑥ Monster reaches up to side
inquisiti
⑦ Monster jumps up to side and crowd starts to run
⑧ Monster grabs person, looks at them and eats them
⑨ Pan out on monster wreaking havoc
sunset
Camera back & away
monster jums down and