

INNOVATIONS

IMAGE BASED 3D EFFECTS

Stefan D'hondt

1302289

supersteffieke@yahoo.co.uk

Table of contents

Abstract	3
Supplied material	3
Aims and objectives	3
Introduction	3-4
Planning and research	4-10
How the different programs work	10-23
Project extensions	23-26
Future program improvement concepts	26-28
Conclusion	29
Appendices	30-37
References	37-38
Acknowledgements	38
Glossary	39-40

Abstract

This project explains the development of several C++ programs which generate 3D image based effects.

Supplied material

- CD-ROM containing the C++ programs.
- VHS tape with sample animations.
- Beta tape with sample animations.

Aims and objectives

Aim

- Create a 3D image-effect using a behavioural system.

Objectives

- Program a standalone OpenGL program that produces a 3D image-effect.
- Produce several animations using the standalone OpenGL program.
- Evaluate the outcome of the program.

Introduction

Computer animation can be looked at from two directions. At one end, we have the artistic side, and at the other end we have the technical aspect to it.

This project explores 3D colour space and behavioural systems as a technical part of computer animation and the images created are there to show the technical abilities of the programs produced throughout this project.

The driving force behind this project is a strong curiosity in procedural animation, behavioural systems and a newfound interest, namely programming.

An initial idea came from the film “X-men”, directed by Bryan Singer. The pintable showing the map of Manhattan, to be more precisely.

This paper will go through the motivations of the author, ideas, illustrate the algorithms used and choices that were made throughout the project.

After this it will handle ideas for extending and improving the current state of the project.

Planning and research

Planning

Before the project was started, some issues needed to be set in stone to avoid going along the wrong route and not being able to finish the project to a state where results could be produced because of time constraints.

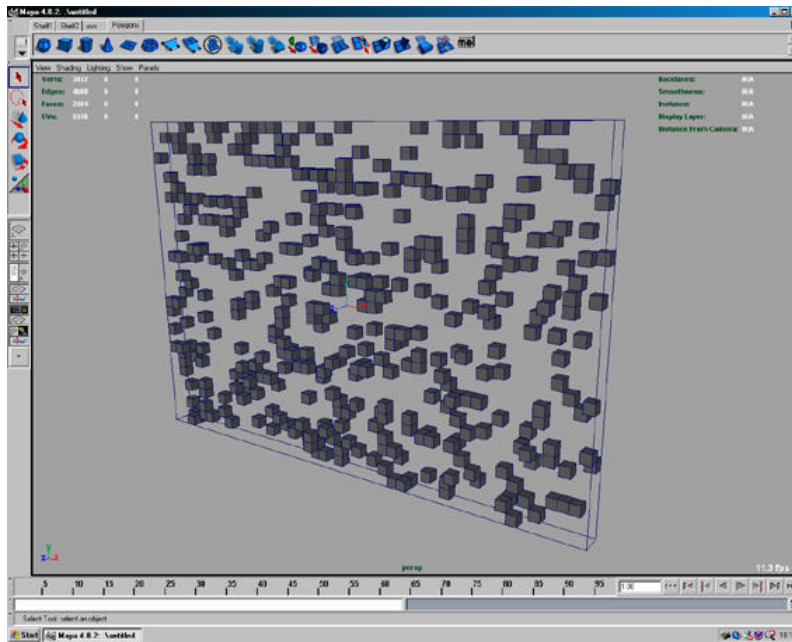
- First major issue was the choice of development environment for producing the project.

Choice of development environment

In the initial stages of the project, three different methods were tried and tested. These methods included SideFX’s Houdini, Alias Wavefront’s MEL-programming language and C++ in conjunction with OpenGL.

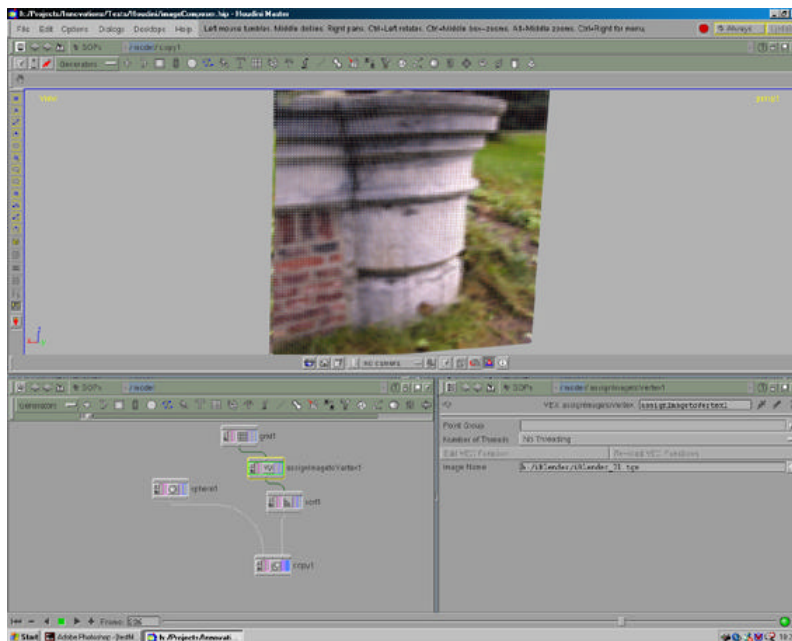
MEL seemed like a good starting point for the project and initial *algorithm*-construction was done in MEL. The benefit of this was that there was an instant visual feedback, so errors or anomalies in *algorithms* were handled straight away.

MEL however, is an *interpreted programming language* which makes it slow to use. Using MEL also has the drawback that it can’t produce standalone executable programs, which C++ can.



The algorithm to scatter the image pixels was first programmed and tested using MEL.

Houdini also seemed like a perfect solution for the set problems. And initial testing was done alongside the algorithm development in MEL. However, after some success, this development path was scrapped because of the limited knowledge the author has about Houdini. If the program was fully understood by the author, this would have been the preferred development environment and would have given more time for creative development.



Initial testing that was done in Houdini consisted of mapping an image to geometry.

Although Houdini is perfectly capable of solving the set problems, C++ in conjunction with OpenGL was used to tackle the problems. C++ is a good choice as a development platform because of its wide user base and extensive reference material.

- Another area of concern was playback smoothness for the final effect produced.

Why write 2 programs?

Procedural animation (handled in section: Planning and research: Research) can be very memory and CPU intensive and thus produce results very slowly.

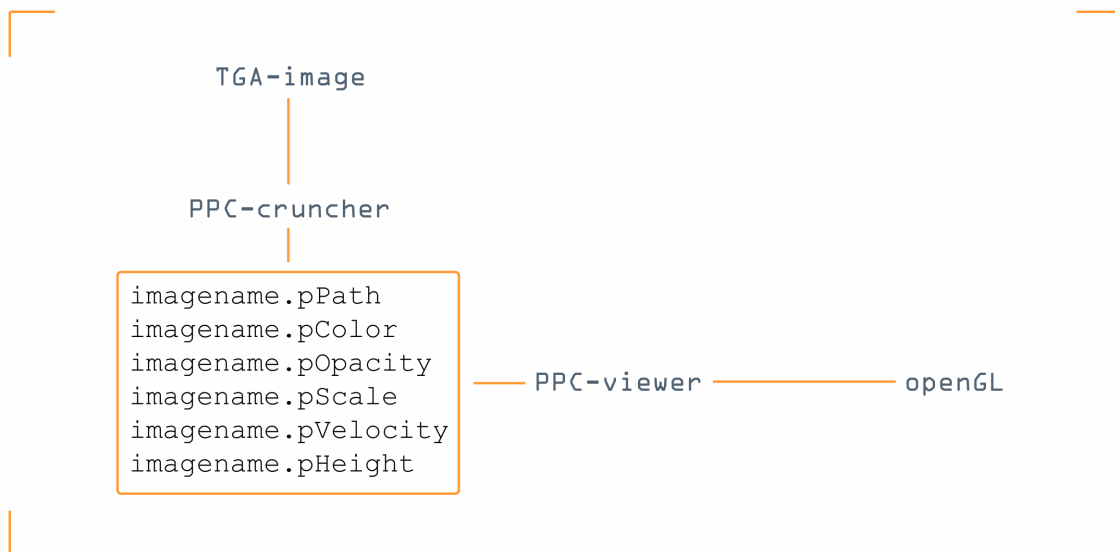
Since the goal of this project is to produce a 3D pixel effect, the aim is to play it back at 25 frames per second.

To achieve this, it was the author's decision to write 2 programs; PPC-cruncher (Pixel Path Constructor cruncher) and PPC-viewer (Pixel Path Constructor viewer).

The first program, PPC-cruncher, reads in a Targa-image, does all the hard work (number crunching) and produces an animation and 6 custom files. (The custom file formats are discussed under topic: PPC-cruncher: What are the files that PPC-cruncher generates? and PPC-cruncher: What data is stored in the custom-created files?)

The second program, PPC-viewer, is able to read in these 6 files, plus an additional RTG-object file and display the animation using OpenGL. (Loading RTG-object files is handled in section: Project extensions: RTG: loading custom-created objects)

Another benefit of this approach is that PPC-cruncher can produce the animations and the user can choose when he wants to view the animation. This way, the user isn't tied down to have to go through all the number crunching every time he wants to look at the animation.



Showing how PPC-cruncher and PPC-viewer, when combined together, form the desired outcome.

Research

Initial research was done to give the author a better knowledge of the different components that make up a behavioural system. Subjects that were covered included: behavioural systems in general, collision avoidance, path finding and artificial intelligence. A major player in this area is Craig W. Reynolds who has been working on this subject for quite a number of years. During this research period, the author was able to start working out general concepts and early testing was done on several algorithms.

What is procedural animation?

In the Merriam-Webster dictionary found on the internet, 'procedural' is explained as:

"a series of instructions for a computer that has a name by which it can be called into action"

What this basically means is that we define a number of instructions for the computer that produces the outcome that we want. If we look at it in terms of computer animation, these instructions will generate an animation as its outcome.

Benefits of procedural animation over keyframe animation:

- Changing a few parameters inside the programming code easily and quickly modifies animation created procedurally.
- If the programming code is constructed with future adaptations in mind and isn't project-specific, code can be reused in other projects.

To solve certain problems in procedural animation, *algorithms* are devised to solve these problems.

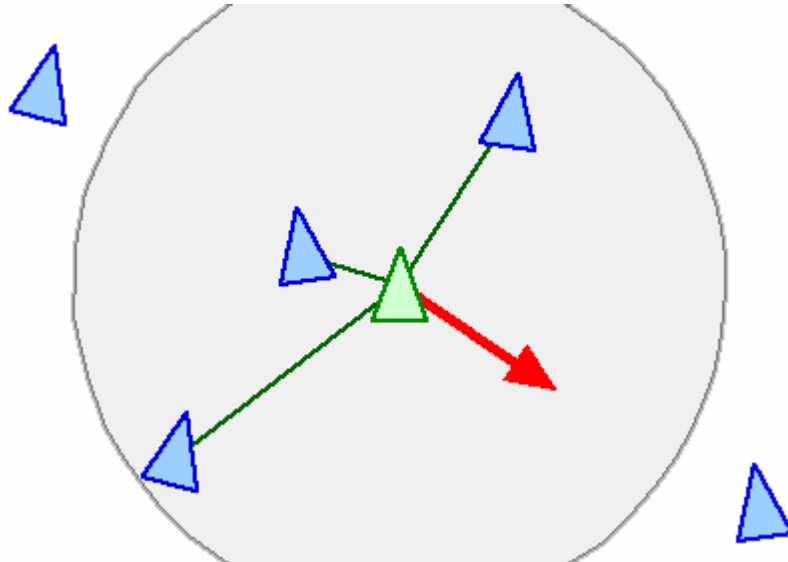
What are behavioural systems and what are they used for?

Behavioural systems are a complex set of instructions working together to mimic certain set behaviours. These behaviours can include *flocking* or *bot navigation* in games.

Although the overall behavioural system can seem very complex, these complex systems can be broken down in very simple rules. Almost all behavioural systems start of from three basic steering behaviours:

- **Separation**

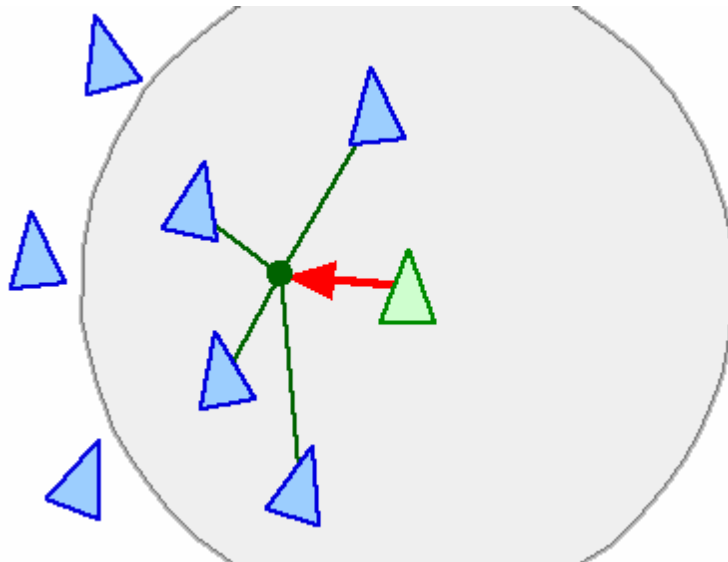
This gives an individual the ability to stay a certain distance away from other characters in the group that are nearby. Separation plays a very important role in collision avoidance on the character interaction level.



A way of implementing separation could be done by checking the direct surroundings for other group characters. For each character found in the direct surrounding, a repulsive force is calculated; these are summed together and added to the individual's acceleration force to produce the final driving force. (Image courtesy of Craig W. Reynolds.)

- **Cohesion**

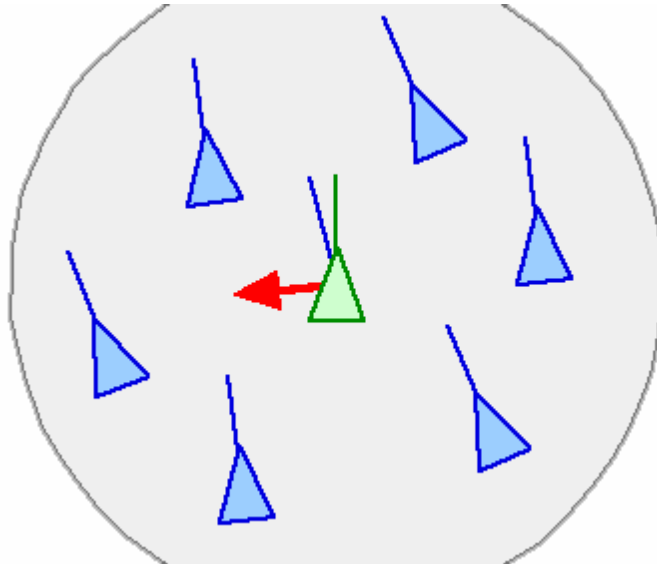
This gives an individual the ability to approach and form a group with other characters in the behavioural system that are nearby.



Cohesion can be implemented by checking the direct neighbourhood of each individual and check for other group members. For the group members found, their average position is calculated. The direction of the current character is adjusted to the average direction of the group. (Image courtesy of Craig W. Reynolds.)

- **Alignment**

This gives an individual the ability to alter its speed and direction to the speed and direction of nearby characters.



Alignment gives the current character the ability to align itself with nearby members of the group. The velocity of the current character is adjusted to the average velocity of nearby group members. (Image courtesy of Craig W. Reynolds.)

Why procedural animation was used in this project?

The author's reasons for using procedural animation are listed in the previous topic. Other reasons are his technical ability and interests as well as being able to quickly visualise different *algorithms*.

How the different programs work

The basic functionality of the project was set to:

- user specifies an image
- program reads in the image
- scatter all the pixels in 3D space
- move all pixels back to their original position within the image

At first, a behavioural system was visualised where an image would be deconstructed to its pixels. These pixels would be randomly spread in 3D space and through the use of a behavioural system, go back to their original position in the image where they belong.

The general use and all options of PPC-cruncher and PPC-viewer are discussed in Appendix A and B accordingly.

PPC-cruncher

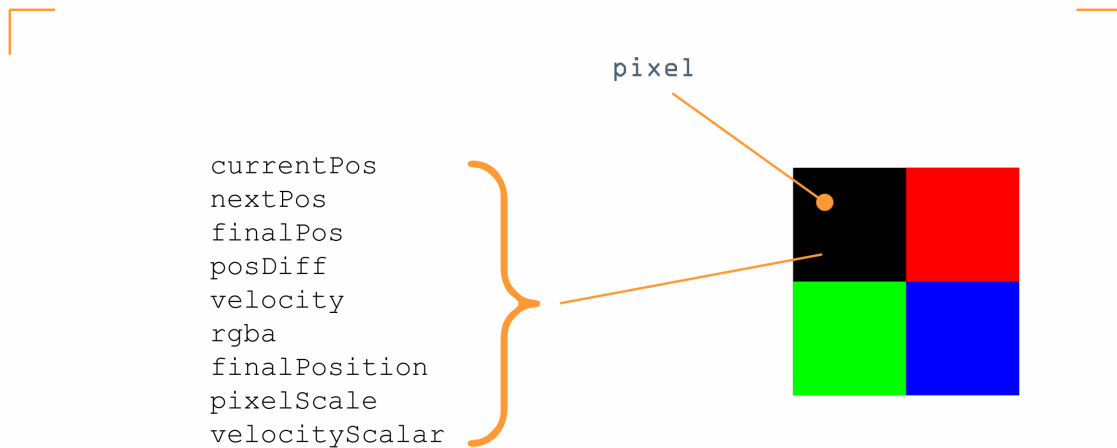
The only essential attribute that needs to be provided to PPC-cruncher is the main TGA-image. This main image is opened and its colour values are stored into a custom-generated pColor-file. (Custom created files are handled in the next topic.) Next, all pixels are created and randomly distributed inside the *scatter volume*. (The scatter-algorithm is discussed later on in the algorithm section.)

By deconstructing an image to its separate pixels, we can define a pixel as the smallest separate entity to work with. By doing this, any number of attributes can be defined and assigned to a single pixel. In this respect, a single pixel can be seen as a particle in a particle system.

Attributes that are assigned to each pixel include:

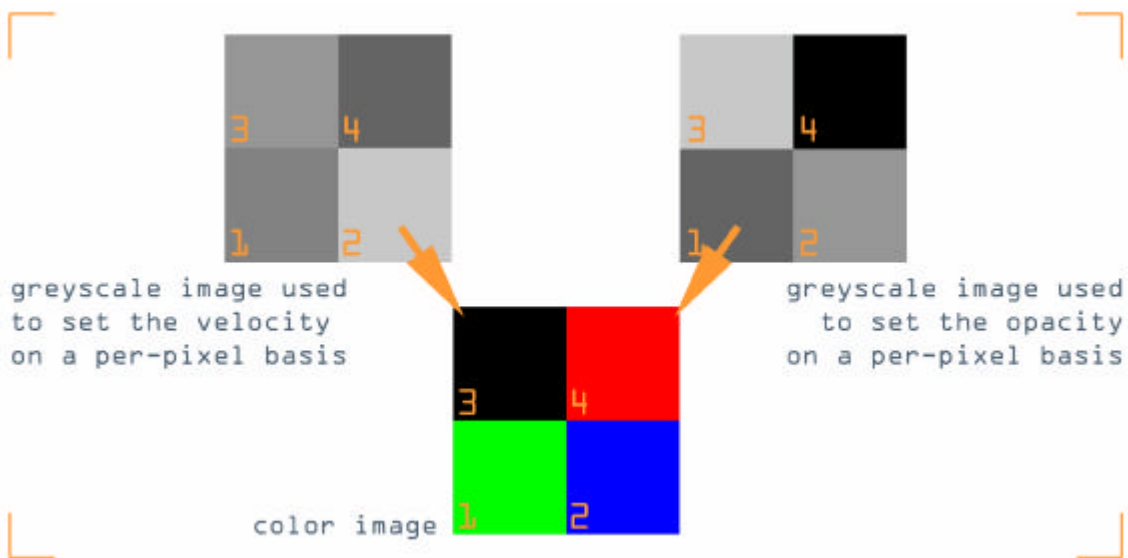
- *currentPos*: current position in 3D space represented with a 3D vector
- *nextPos*: next position in 3D space represented with a 3D vector
- *finalPos*: in 3D space represented with a 3D vector
- *posDiff*: describing the difference of the current and final position represented as a 3D vector
- *velocity*: the pixels' velocity represented as a 3D vector
- *rgba*: a custom storage class to hold the red, green, blue and alpha value of the pixel
- *finalPosition*: a Boolean value that shows if the pixel is at its final position or not
- *pixelScale*: a scalar for the size of the pixel represented as a fractional value

- *velocityScalar*: a scalar for the velocity of the pixel represented as a fractional value



This figure illustrates all the different attributes that are attached to a single pixel.

Following this line of thought, it was the choice of the author to include the ability to read in separate greyscale Targa-images of the same size as the original colour image. These greyscale colour values can then be assigned to different attributes of a single pixel. This extensively extends the visual possibilities of the project.



This shows the idea behind assigning greyscale images to an attribute of a pixel on a per-pixel basis. Pixel 1 of the top left greyscale image will be assigned to the velocity of pixel 1 of the image in the centre. Pixel 1 of the top right greyscale image will be assigned to the opacity of pixel 1 of the image in the centre. This will be done for all the pixels in the main image.

The next task is getting to move the pixels back to their final position in the image and so creating the actual animation. This animation is saved in a file: *imagename.pPath*. (Custom created files are handled in the next topic.)

What are the files that PPC-cruncher generates?

PPC-cruncher initially created only 2 files:

- *imagename.pPath*
- *imagename.pColor*

Throughout the development of the program, four extra files were written out:

- *imagename.pOpacity*
- *imagename.pScale*
- *imagename.pHeight*
- *imagename.pVelocity*

All custom files are saved in the *ASCII* format.

What data is stored in the custom-created files?

pPath

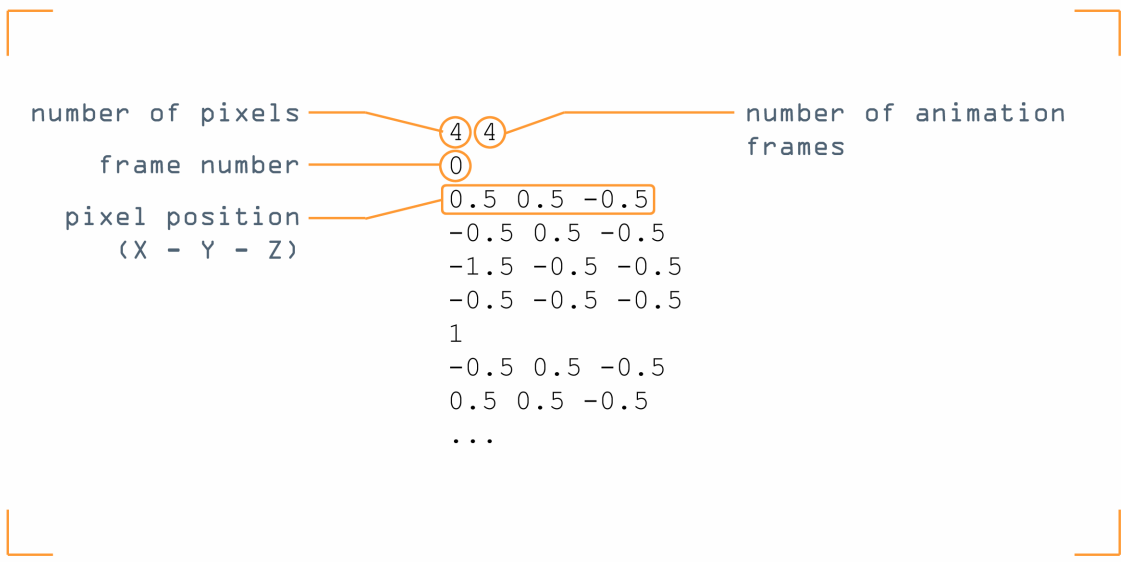
This file is critical to the project. It stores all the frames of the generated animation for all the pixels.

- The first number stands for the number of pixels in the image.
- The second number stands for the number of animation frames.
- Then each animation frame is described as its frame number and the position of all the pixels in the image as 3 numbers representing its x, y and z values correspondingly.

```

0
0.5 0.5 -0.5
-0.5 0.5 -0.5
-1.5 -0.5 -0.5
-0.5 -0.5 -0.5
1
-0.5 0.5 -0.5
0.5 0.5 -0.5
-0.5 -0.5 -0.5
0.5 -0.5 -0.5
2
-0.5 -0.5 -0.5
0.5 -0.5 -0.5
-0.5 0.5 -0.5
0.5 0.5 -0.5
3
-0.5 -0.5 0
0.5 -0.5 0
-0.5 0.5 0
0.5 0.5 0

```



This shows how the custom-created pPath animation file is structured.

pColor

- The first number stands for the number of pixels in the image.

- The colour of each pixel is stored as 4 whole numbers representing Red, Green, Blue and Alpha respectively ranging between 0 and 255.

```
4
0 0 255 255
255 0 0 255
0 255 0 255
127 127 127 255
```

pOpacity

- The first number stands for the number of pixels in the image.
- The opacity of each pixel is stored as a whole number with a range between 0 and 255.

```
4
255
127
100
0
```

pScale

- The first number stands for the number of pixels in the image.
- Every new line represents a uniform pixel size scalar.

```
4
1.6
1.2
0.8
0.4
```

pVelocity

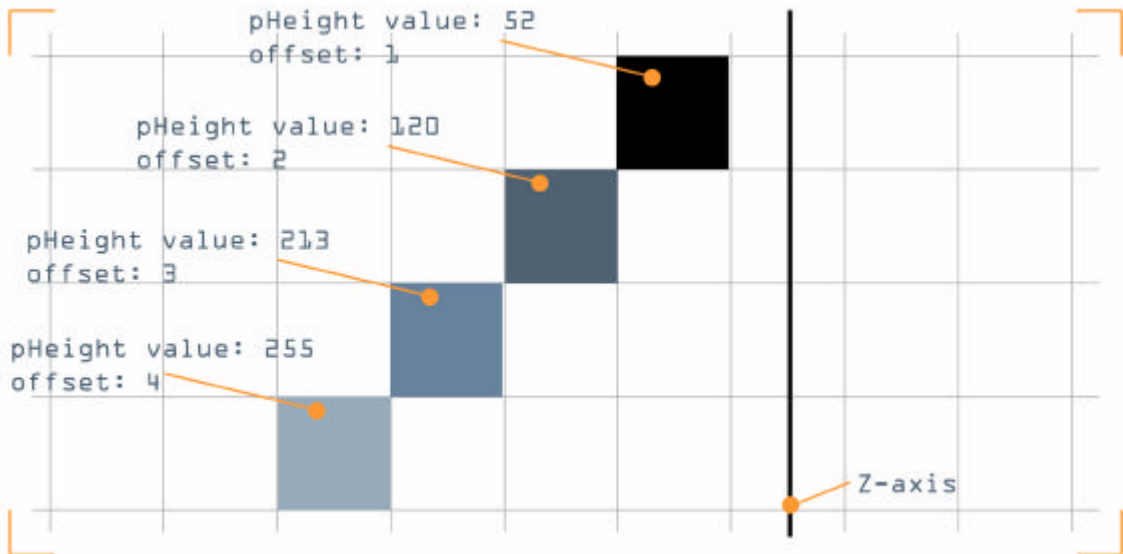
- The first number stands for the number of pixels in the image.
- Every new line represents a pixel velocity scalar.

```
4
1
1
1
1
1
```

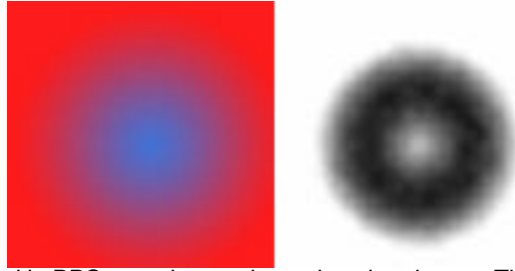
pHeight

- The first number stands for the number of pixels in the image.
- Every new line represents a pixel height offset. The offset will be higher when the greyscale value gets closer to 255.

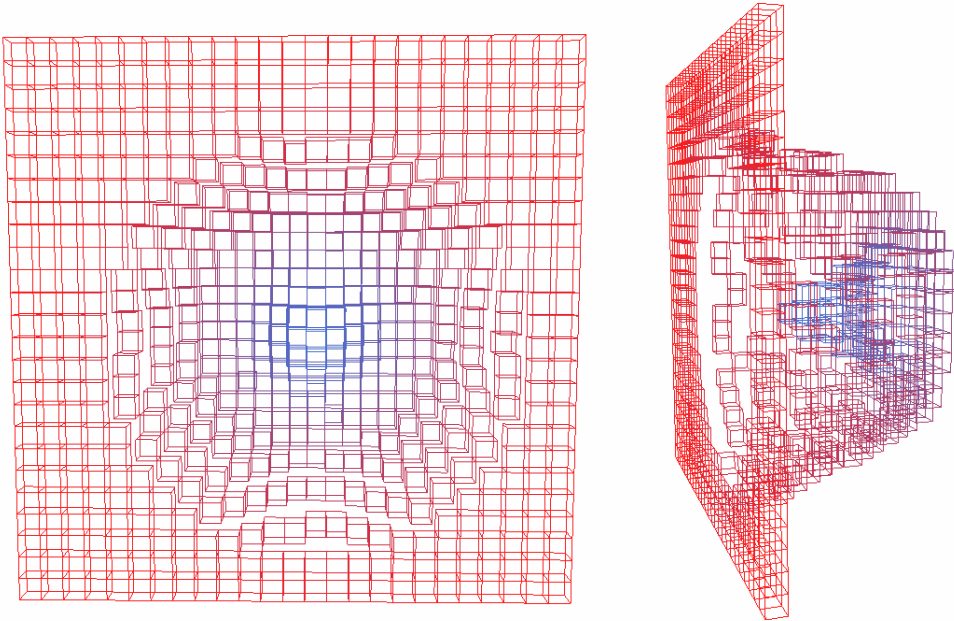
```
4
52
120
213
255
```



This figure shows the principle of a greyscale image mapped as a Z-offset to a pixels' position. The higher the greyscale value, the higher the offset will be. The greyscale values will be converted to the range 0–maxOffset as an integer.



The image on the left is used in PPC-cruncher as the main colour image. The image on the right is the greyscale height map used on the colour image to offset bright pixels.



The above two images show the effect in 3D-space on the original colour image.

PPC-cruncher algorithms

- **Scattering the pixels: algorithm outline**

```
for every pixel in the image
{
    generate random position in the 3D scatter volume
}

assume that there are equal positions for some pixels

for every pixel in the image
{
    check current pixel position against all other pixel positions
```

```

    if there is an equal position
    {
        generate new random position in 3D scatter volume
        increment a counter
    }
}

if the counter is not 0
{
    reset counter
    repeat the previous step
}

if the counter is 0
{
    finished
}

```

A *scatter volume* is used as a boundary for the pixels. There are two good reasons why the author chose to do this:

- The whereabouts of the scattered pixels can be semi-controlled. Meaning that the pixels are still randomly distributed but restrictions can be set.
- Reducing the size of the *scatter volume* has the result that the pixels' random distribution will be closer to their final destination, and thus resulting in less animation frames that are generated. On the other hand, increasing the *scatter volume* will distribute the pixels more freely and thus result in more animation frames that are generated.

Current algorithm issues:

- When larger images are specified to work on, the time it takes to check all the pixel positions for equality increases exponentially.

- It is perfectly possible that the random positions generated are truly different from each other so unnecessary checking is done afterwards.
- Because 3 relatively small random numbers are generated, one for each position component of the 3D vector, it is more likely that equal numbers are generated.

All these issues raised here will increase the execution time of the program.

- **Moving pixels around: algorithm outline**

```

while all the pixels aren't at their final position
{
    for every pixel in the image
    {
        check if posDiff is 0

        if posDiff is not 0
        {
            -get the highest absolute component value of the
              posDiff vector (<<-4, 3, 0>> would result in x)

            -set the velocity vector to 1 for the highest
              component and 0 for the other 2 components
              (velocity vector would be <<1, 0, 0>>)

            -check if the current pixel position is to the left,
              right or below or above its final position and set
              the sign accordingly

            -multiply the velocity vector with the velocity
              scalar (velocity vector would be <<velocity scalar,
              0, 0>>)

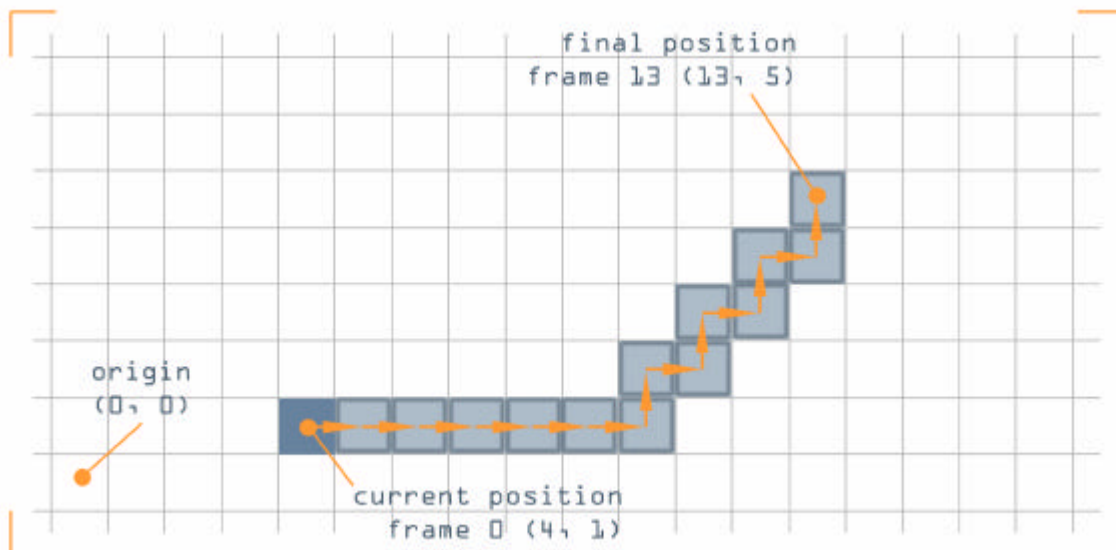
            -set the nextPos vector to (currentPos + velocity)
        }
    }
}

```

```
if posDiff is 0
{
    set nextPos to currentPos
}
}
```

In the case of several posDiff components to be equal, then the X-component will get priority over the Y-component, and the Y-component over the Z-component.

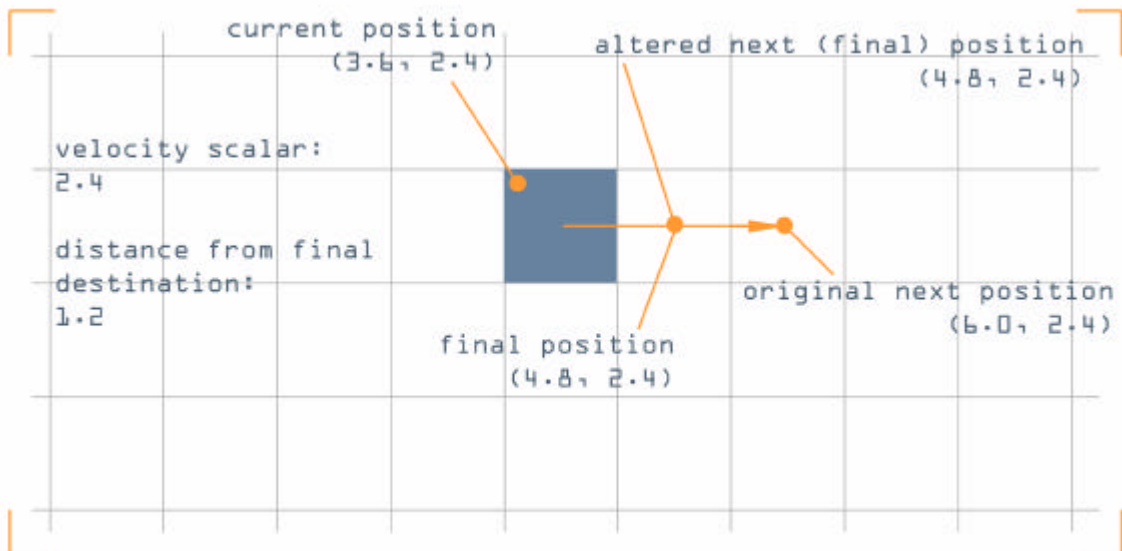
Implementing this algorithm will result in a stepped effect when the pixels are moving.



This figure illustrates the stepped effect produced by the algorithm that moves the pixels around in 3D space. For illustration purposes and clarity, the algorithm is shown here working on a 2D case.

There is however some checking to be done to avoid running into infinite loops.

Consider this case:

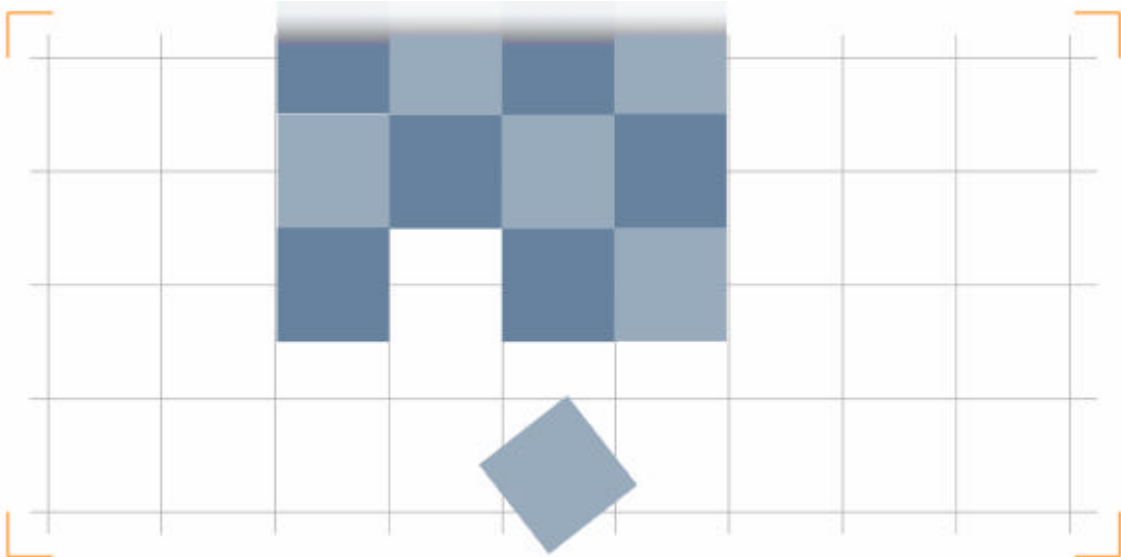


This figure demonstrates how an infinite loop can be circumvented by altering a pixels' velocity so it can reach its final destination.

- **Why pixels don't rotate**

Pixel distribution in images can be considered as several rows and columns of pixels fitted together to form the actual image. This has a result that every pixel has four other pixels surrounding it, thus sharing its four edge positions with other pixels (except in the case of pixels lying on the outer edge of the image).

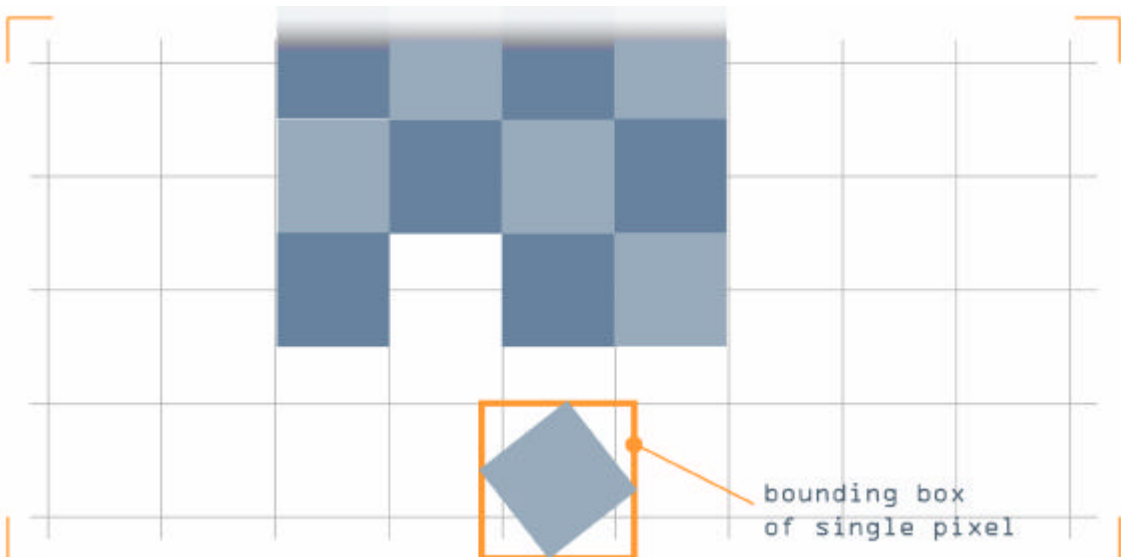
Now consider this case:



When deciding if pixels should be able to rotate, a case like this where only 1 pixel isn't at its final destination yet, might be common.

At first sight, there doesn't seem to be a problem, but at closer examination, problems would appear when implementing collision avoidance based on the *bounding box* of a pixel.

Implementing collision avoidance based on the *bounding box* of a pixel works fine if the pixel is rotated at right angles to the world axis. As soon as it is at a non-right angle, the *bounding box* grows to fit the object.



Now the bounding box of the pixel is highlighted, the problem that this case causes when collision avoidance is implemented becomes clear. The dimensions of the bounding box are too big to fit in the space available.

Implementing collision avoidance this way would prevent pixels from reaching their final destination.

- **Loading images: the TGA-loader**

A TGA-loader was downloaded to give a head start and let the author focus more on writing the actual programs. This can be justified with a simple reason. Although the project was based on reading in TGA-images, the aim was to produce a 3D-effect. So at the start, the main focus was to get results quickly without worrying about how to read in image files.

PPC-viewer

The main purpose of PPC-viewer is to view the animation generated with PPC-cruncher.

What PPC-viewer does is present the user with a number of options. These options include which of the six custom-files to load. The user can choose just to specify one or three or all six of them. The only essential file that needs specifying is the main animation file. (pPath)

If no other files are specified, default numeric values will be used.

After this, all the specified files are stored in the memory of the computer and the OpenGL display will be created.

Project extensions

RTG: loading custom-created objects

Early on in the project, individual pixels were represented as cubes. This was fine for the development stage since the focus was to get the programs to a state that satisfied the initial goals that were set. Although cubes are perfectly fine as a representation for pixels, the author wanted to give the user a choice as to how to display the pixels.

Although a number of file formats could have been picked, RTG was preferred. The RTG-format has a wide range of possibilities, easy accessibility, readability and it has its own export-function inside Maya with several useful options. Not to mention, future value for different projects.

iBlender

iBlender is a small program derived from PPC-cruncher and PPC-viewer. This program leans more toward the initial pintable idea from the film X-men.

What does it do?

iBlender reads in a number of TGA-images, and blends between those images in a set number of interpolation frames. It does this however in a 3D-fashion and could be seen as an animated displacement map.



Generated interpolation frame inside iBlender where each pixel of the image is extruded according to its luminance value.

How does it work?

The user has to supply iBlender with a text file that holds vital information. A simple iBlender animation file might look like this:

```
#iBLENDER_CONFIGURATION_FILE

#NUMBER_OF_KEYFRAMES 4

#MINDISPLACEMENT 0.0
#MAXDISPLACEMENT 75.0

#IMAGEPATH c:\\iBlender\\iBlender_01.tga
#NUMBER_OF_INTERPOLATION_FRAMES 40

#IMAGEPATH c:\\iBlender\\iBlender_02.tga
#NUMBER_OF_INTERPOLATION_FRAMES 60

#IMAGEPATH c:\\iBlender\\iBlender_03.tga
#NUMBER_OF_INTERPOLATION_FRAMES 40

#IMAGEPATH c:\\iBlender\\iBlender_04.tga
#NUMBER_OF_INTERPOLATION_FRAMES 60
```

A configuration file speaks for itself:

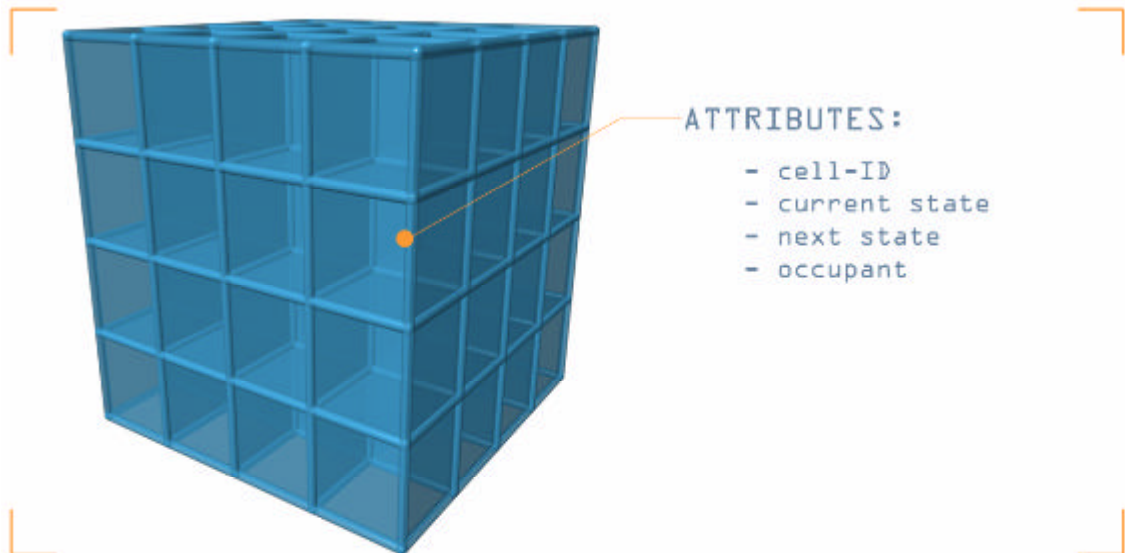
- The first command that every iBlender configuration file needs is `#iBLENDER_CONFIGURATION_FILE`. This will let the program know that what follows is a description of an animation.
- The second command has to be `#NUMBER_OF_KEYFRAMES`, followed by an integer number stating how many images the user will supply the program with.
- Next are `#MINDISPLACEMENT` and `#MAXDISPLACEMENT`, both followed by a fractional number to state the minimum and maximum displacement values for the pixels.

- Then, for the defined number of keyframes, the user specifies the absolute paths to the images and an integer number representing the number of interpolation frames between each image.

When such a configuration file is specified inside iBlender, it will then generate and store the interpolation frames in computer memory and play the animation in an OpenGL window.

Future program improvement concepts

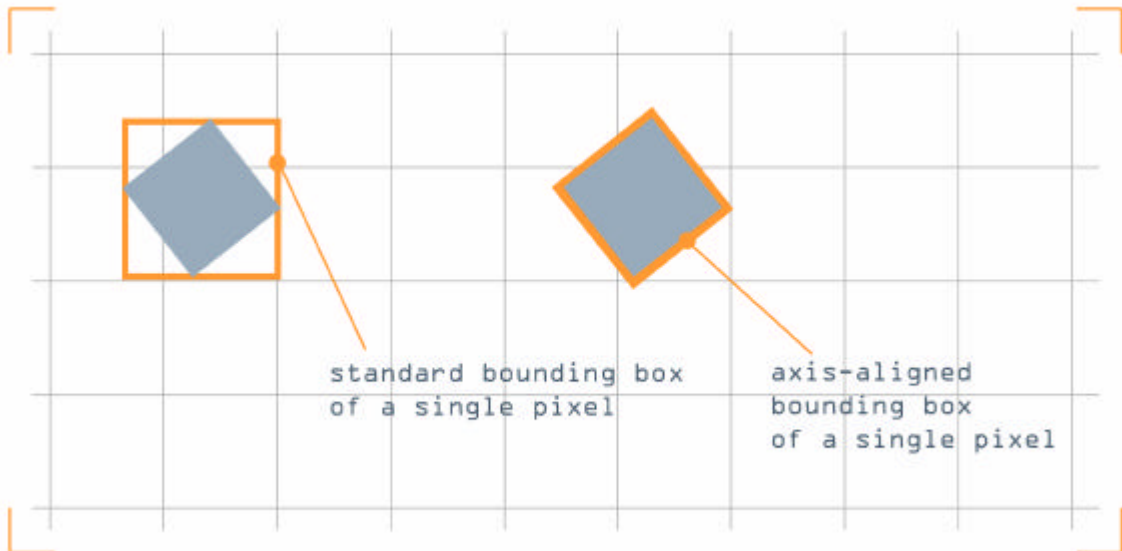
- The *scatter volume* can be put to more use by dividing it up in segments like a chessboard, resulting in a 3D grid. Each grid cell would have several attributes assigned to it. These would include:
 - *cellId*: Unique number that identifies each grid cell represented as an integer number.
 - *currentState*: Boolean value that represents if the cell is occupied.
 - *nextState*: Boolean value that represents if the cell will be occupied the next frame.
 - *occupant*: The unique pixel number that currently occupies the grid cell or will occupy the grid cell in the next frame represented by an integer number.



Listing the different attributes that could be assigned to a single scatter volume cell.

Dividing the *scatter volume* up into grid cells has two major implications:

- Collision avoidance can be implemented more easily. A pixel can just check the `currentState` and `nextState` of the grid cell that it wants to occupy next.
- Finding a scatter position for every pixel means that only one random number has to be generated in the range between 0 and the maximum number of grid cells. There is no longer any need to assume that there are equal pixel positions since the randomly generate number can be checked against the according grid cells' `currentState` variable.
- To implement collision avoidance, there can be opted to use axis-aligned *bounding boxes*. What this means is that bounding boxes will rotate with the object they enclose.



Showing the difference between a standard bounding box versus an axis-aligned bounding box.

This would mean that the *bounding box*'s dimensions would stay the same, even when the object that it encloses rotates along any of the three axes.

- When working on large images, program execution will be fairly long. There are several options to reduce the running length of the program:
 - The velocity scalar can be increased.
 - The *scatter volume* size can be kept to a minimum, thus reducing the distance between the scatter position of a pixel and its final position. This, however, increases the execution length of the scatter-algorithm.

A solution to this would be interpolation the animation frames when viewing the generated animation.

This has the advantage that we can increase the velocity scalar, keep the *scatter volume* size quite big and decrease the final .pPath animation file size. The outcome of this would be that less animation frames are generated during number crunching time, creating a more jumpy animation. This jumpy animation is circumvented by interpolating each generated animation frame with several interpolation frames generated on the fly while viewing the animation.

Conclusion:

I believe that the 3 programs produced meet the initial goal set. However, collision avoidance was on my to-do list and I didn't get around implementing it. I do not feel that, after viewing some animation produced by the programs, this missing feature plays a big role in the visuals produced.

The programs produced might not have a high wow-factor but it was a good learning lesson in terms of programming and maths.

And although, PPC-cruncher and PPC-viewer were the main focus of the project, I am more satisfied with the effect that is generated with iBlender.

The project was a joy to work on and thus programming could run into the early morning hours which I think shows the interest and fascination for the chosen subject.

Appendices

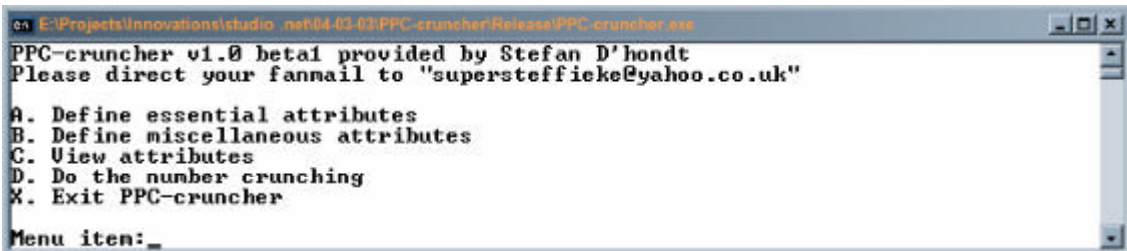
Appendix A: PPC-cruncher: how to use the program

Note:

- No spaces can be used when defining a path to an image or RTG object file.
- For every backslash used inside a path declaration, a second one has to be added.
- All explanations of options assume that the user is currently inside the main menu.
- Program menu items and examples are displayed in *Courier New*, italics, size 10.

The only essential attribute that needs to be defined is a colour image for PPC-cruncher to work on. All the other options the user is able to define aren't essential for the program to work properly.

When starting PPC-cruncher, the user is presented with the main menu.



```
E:\Projects\Innovations\studio_net04-03-03\PPC-cruncher\Release\PPC-cruncher.exe
PPC-cruncher v1.0 beta1 provided by Stefan D'hondt
Please direct your fanmail to "supersteffie@yaho.co.uk"

A. Define essential attributes
B. Define miscellaneous attributes
C. View attributes
D. Do the number crunching
X. Exit PPC-cruncher

Menu item: _
```

The main menu for PPC-cruncher.



```
E:\Projects\Innovations\studio_net04-03-03\PPC-cruncher\Release\PPC-cruncher.exe
Menu item:a

A. Define color map
B. Define pixel scatter volume scalar
X. Show main menu

Menu item: _
```

The "essential attributes" options.

First, the user defines a colour image for PPC-cruncher to work with.

To define a colour image:

- A. Define essential attributes
- A. Define color map

- *Enter the absolute path to the image.*

Example: `c:\PPC-cruncher\image.tga`

To define the maximum volume that pixels can be scattered in, the scatter volume is used.

To alter the size of the *scatter volume*:

- *A. Define essential attributes*
- *B. Define pixel scatter volume scalar*
- *The user is then asked to enter 3 integer numbers; one for every direction.*



The "miscellaneous attributes" options.

Several options can be set under the miscellaneous attributes menu.

To enter the miscellaneous attributes menu:

- *B. Define miscellaneous attributes*

Any greyscale TGA-image, that is the same size as the main colour image, can be defined in any of the following options:

- *A. Define opacity map*
- *C. Define velocity scalar map*
- *E. Define pixel scalar map*
- *G. Define height displacement map*

Example: `c:\PPC-cruncher\imageGreyscale.tga`

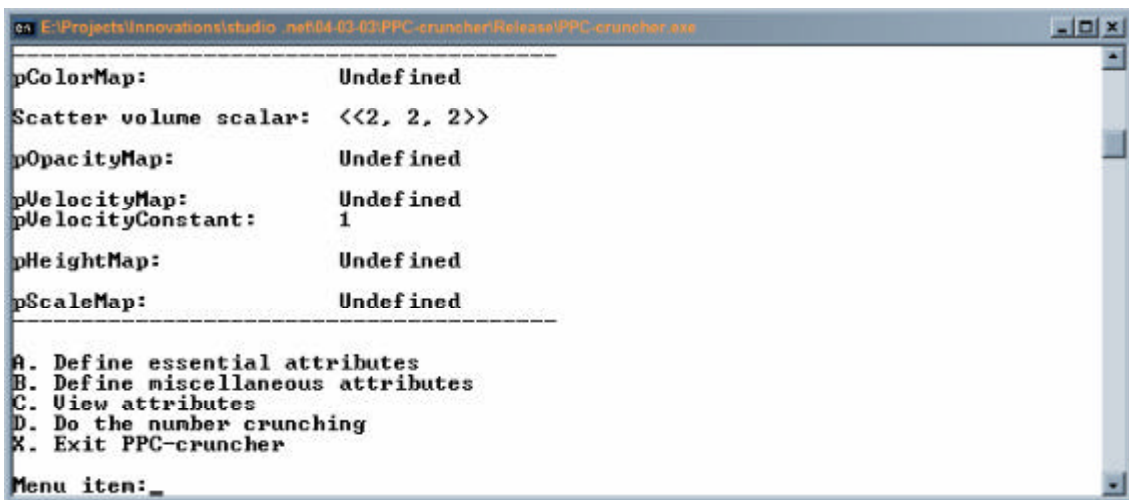
A few of these options can also be set by defining scalar values instead of greyscale images. The options that support this:

- *B. Define opacity constant*
- *D. Define velocity scalar constant*
- *F. Define pixel scalar constant*

Each of these scalars have a range between 0 and 255.

Note:

- If both a greyscale image and a scalar value are provided for a single option, then only the defined image is considered by PPC-cruncher.

A screenshot of a Windows application window titled "E:\Projects\innovations\studio_nsf04-03-03\PPC-cruncher\Release\PPC-cruncher.exe". The window displays a text-based interface with the following content:

```
pColorMap:          Undefined
Scatter volume scalar: <<2, 2, 2>>
pOpacityMap:        Undefined
pVelocityMap:       Undefined
pVelocityConstant:  1
pHeightMap:         Undefined
pScaleMap:          Undefined
-----
A. Define essential attributes
B. Define miscellaneous attributes
C. View attributes
D. Do the number crunching
X. Exit PPC-cruncher
Menu item: _
```

The "view attributes" menu, displaying the current settings of the attributes.

At any time, the user has a choice to view the attributes it has already entered by going to menu:

- *C. View attributes*

Note:

- If needed, options can be changed at all times, just by navigating to the appropriate attribute and redefining it.

When all the necessary options are set; calculating the animation is the next step.

This can be done by going to menu:

- *D. Do the number crunching*
- Confirm by pressing *y* or *Y*

PPC-cruncher will then read all the specified images, read in any necessary scalar values and calculate the animation.

The six files that are generated by PPC-cruncher are stored on the hard disk in the same directory as the images.

Appendix B: PPC-viewer: how to use the program

Note:

- No spaces can be used when defining a path to an image or RTG object file.
- For every backslash used inside a path declaration, a second one has to be added.
- All explanations of options assume that the user is currently inside the main menu.
- Program menu items and examples are displayed in *Courier New*, italics, size 10.

The only essential attribute that needs to be defined is a pPath animation file for PPC-viewer to work on. All the other options the user is able to define aren't essential for the program to work properly.

When starting PPC-viewer, the user is presented with the main menu. Here he or she can define all the different files for PPC-viewer to load in.

```

E:\PPC-viewer.exe
RTG: Creating RTG-object instance...done
PPC-cruncher v1.0 beta1 provided by Stefan D'hondt
Please direct your fanmail to "supersteffieke@yahoo.co.uk"

A. Define pPath-file
B. Define custom object file. <RTG-file>
C. Define pColor-file
D. Define pOpacity-file
E. Define pScale-file
F. View attribute files
G. View animation
X. Exit PPC-viewer

Menu item:

```

The main menu from PPC-viewer.

To define the only truly necessary file, the user has to navigate to:

- A. Define pPath-file

Example: If the main image would have been defined as: `c:\PPC-cruncher\imagename.tga`, the pPath file can be defined as: `c:\PPC-cruncher\imagename.tga.pPath`

After this, the user has a choice of specifying any of the following files:

- RTG: `c:\PPC-cruncher\customShape.rtg`
- pColor: `c:\PPC-cruncher\imagename.tga.pColor`
- pOpacity: `c:\PPC-cruncher\imagename.tga.pOpacity`
- pScale: `c:\PPC-cruncher\imagename.tga.pScale`

```

E:\PPC-viewer.exe
-----
pPathFile:           Undefined
RTG-file:           Undefined
pColorFile:         Undefined
pOpacityFile:       Undefined
pScaleFile:         Undefined
-----
A. Define pPath-file
B. Define custom object file. <RTG-file>
C. Define pColor-file
D. Define pOpacity-file
E. Define pScale-file
F. View attribute files
G. View animation
X. Exit PPC-viewer

Menu item:_

```

The "view attributes" menu, displaying the current settings of the attributes.

At any time, the user has a choice to view the attributes it has already entered by going to menu:

- *F. View attributes*

Note:

- If needed, options can be changed at all times, just by navigating to the appropriate attribute and redefining it.

To view the animation, the user can navigate to:

- *G. View animation*

OpenGL keyboard and mouse commands:

- left mouse button: rotate
- middle mouse button: zoom in / out
- right mouse button: move camera

- 'a' or 'A': anti-aliasing on / off
- 'd' or 'D': change display mode between wireframe / solid / solid + wireframe
- 'f' or 'F': reset camera to its original settings
- 'g' or 'G': grid on / off
- 't' or 'T': transparency on / off
- ',' or '<': play backwards when playback is started / step to previous frame when playback is stopped
- '.' or '>': play forwards when playback is started / step to next frame when playback is stopped
- <SPACE>: playback start / stop
- <ESC>: exit

Appendix C: iBlender

Note:

- No spaces can be used when defining a path to an image or RTG object file.
- For every backslash used inside a path declaration, a second one has to be added.


```
E:\iBlender.exe
iBlender configuration file:e:\\iBlender.conf
-----
Image:                e:\\iBlender\\iBlender_31.tga
Image size:           75 x 100
# of pixels:          7500
Pixel depth:          24
# of channels:         3
-----
Image:                e:\\iBlender\\iBlender_32.tga
Image size:           75 x 100
# of pixels:          7500
Pixel depth:          24
# of channels:         3
-----
Image:                e:\\iBlender\\iBlender_34.tga
Image size:           75 x 100
# of pixels:          7500
Pixel depth:          24
# of channels:         3
-----
A. Specify an iBlender configuration file.
B. View animation.
X. Exit iBlender.
Menu item: _
```

This is the menu presented after the user specified a valid iBlender configuration file.

openGL keyboard and mouse commands:

- left mouse button: rotate
- middle mouse button: zoom in / out
- right mouse button: move camera

- 'd' or 'D': change display mode between wireframe / solid / solid + wireframe
- 'f' or 'F': reset camera to its original settings
- 'g' or 'G': grid on / off
- 't' or 'T': transparency on / off
- <SPACE>: playback start / stop
- <ESC>: exit

References

- Craig W. Reynolds: www.red3d.com
- Game and algorithm design: www.gamasutra.com
- Targa loader: Lev Povalahev: www.levp.de/3d/index.html
- Merriam-Webster dictionary: www.m-w.com

- 'OpenGL Programming guide: The Official Guide to learning OpenGL', Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner
- 'Teach yourself C++ in 10 minutes', 2002, Jesse Liberty, Sams
- 'Teach yourself C++ in 24 hours', 2002, Jesse Liberty, Sams
- 'Aan de slag met C++', 2000, Gertjan Laan, Academic service
- 'Introduction to data structures and algorithm analysis with C++', 1995, George J. Pothering and Thomas L. Naps

Acknowledgements

I would like to thank the following people for their contribution to the project:

Steve Bell – Rob Bateman – Adam Vanner – Erwin D'hondt

Glossary

Algorithm

A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation; a step-by-step procedure for solving a problem or accomplishing some end especially by a computer.

ASCII

Stands for American Standard Code for Information Interchange.

Bot navigation

A bot (robot) is a computer controlled character in a game. Bot navigation therefore is a means of bots to move around in a set environment without the need for human interaction.

Bounding box

The bounding box of an object is the smallest possible box volume, which entirely encloses the object.

Entity

Independent, separate, or self-contained existence.

The existence of a thing as contrasted with its attributes.

Something that has separate and distinct existence and objective or conceptual reality.

Flocking behaviour

Group behaviour where each individual of the group takes the overall group behaviour into account and behaves accordingly. An example where flocking is used is crowd simulation.

Interpreted programming language

An interpreted programming language doesn't compile any code before execution, but instead works through the code (or interprets) the code during execution time. This has the disadvantage that program execution will go considerably slower than code that was compiled before program execution.

Particle

A relatively small or the smallest discrete portion or amount of something.

Pixel

Any of the small discrete elements that together constitute an image. (as on a television screen)

Procedure

A series of instructions for a computer that has a name by which it can be called into action.

Scatter volume

An invisible 3D box, acting as a boundary wherein the pixels are randomly placed, and restricting the pixels to go outside this volume.