



DEV.MAG

CREATE • DEVELOP • EXPERIENCE

BACK OF A NAPKIN:

PART 3: WHAT IS TEXTURE FILTERING

THE WORK

AFTER THE WORK

THE BASICS

OF SOUND

DEVELOPMENT

MOBILE GAMES:

PART 2: MAKING MOVES

REVIEWS : GAUNTLETS FEATURE :
ENGINES AND FRAMEWORKS,
THE QUALITY TOUCH



DEV.MAG ISSUE 4

SOUTH AFRICA'S FIRST GAME DEVELOPMENT MAGAZINE

Crash Bandicoot 2D

CONTENTS

REGULARS

03 - ED'S NOTE

04 - DIGITAL STOMPIES

FEATURE

05 - ENGINES AND FRAMEWORKS

SPOTLIGHT

07 - Unc1354m

REVIEW

08 - GAUNTLETS

DESIGN

09 - THE WORK AFTER THE WORK

10 - BACK OF A NAPKIN PART 3:
WHAT IS TEXTURE FILTERING?

13 - MAKING 2D ASSETS WITH 3D
SOFTWARE

14 - THE QUALITY TOUCH PART 1:
THE MINIMUM REQUIREMENTS

15 - SOUNDS GOOD: THE BASICS OF
SOUND DEVELOPMENT

TECH

17 - A LITTLE BIT ABOUT RECURSION

MOBILE

18 - MOBILE GAME DEVELOPMENT
IN JAVA

TAILPIECE

20 - THE TRUTH ABOUT
INSTITUTIONS



08



10



09

ED'S NOTE

Something has really grabbed the attention of the guys at Dev.Mag and I, and that is just how far this project has come. Every week I hear stories and see the evidence of what we have accomplished. We've been mentioned in NAG (a gaming magazine that gets distributed internationally) and we are also a regular feature on their monthly cover DVD. We've been mentioned in a newspaper (The Citizen), in blogs, and in numerous other places.

This is our 4th issue, which is, apparently, the 'life-defining' edition of a magazine. I quote a GameMaker forum member: "Most e-zines don't get to their 4th edition, they get their 3rd out but never a 4th. There's some sort of invisible barrier there or something. The ones that make it though are usually here to stay."

If our increasing quality is anything to go by, we'll be here for quite a while. But besides all the promotional talk, I've also got some other news: May/June saw the release of another game development e-magazine, GameForce. We were lucky enough to be sent a complementary copy (+3 points) by the staff. All I can really say to them is "good luck and I hope you break the 4th edition barrier".

Editor
Stuart 'GoNz0' Botma



THE TEAM

ENIGMATIC ED

Stuart "GoNz0" Botma

DASTARDLY DEPUTY

Rodain "Nandrew" Joubert

DILIGENT DESIGNERS

Brandon "CyberNinja" Rajkumar
Paul "Higushi" Myburgh

JUBILANT JOURNALISTS

Simon "Tr00jg" de la Rouviere
Ricky "Insomniac" Abell
William "cairswm" Cairns
Bernard "BurnAbis" Boshoff
Danny "dislekcia" Day
Andre "Fengol" Odendaal
Yuri "knet" Oyoko
Heinrich "Himmler" Rall
Matt "Flint" Benic
Luke "Coolhand" Lamothe
Greg "Zphyr" Reveret

WIZARDLY WEBSTER

Claudio "Ch1ppit" de Sa

WEBSITE

devmag.googlepages.com

To join, make suggestions or just tell us we're great, contact:
devmag@gmail.com

This magazine is a project of the NAG Game.Dev forum. Visit us at
www.nag.co.za

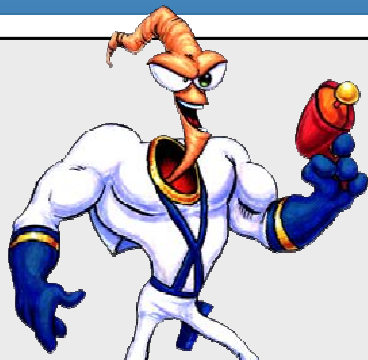


NEW LOCAL SITE

A prolific game developer in South Africa has recently launched their own Game Development site (www.TheGameDeveloper.co.za). Much like a blog, TheGameDeveloper will give voice to ideas and methods of game development, with a focus on things such as frameworks and Delphi.

The Game Developer

The Game Developer



DEVELOPMENT GOLDMINE?

Need art? Need sound? Need resources? Need *anything* for a game and don't want to go to the hassle of creating it yourself? Try the Game Contents Resources from gpwiki.org, a place with a lot of cool links to online galleries, gaming material and a whole manner of resources including sound, music, 2D/3D art, editors, icons and more. Best of all, it's *free*. Get it all now at http://gpwiki.org/index.php/Game_Content_Resources.

Contents [hide]

- 1 Art Resources
 - 1.1 3D
 - 1.2 Fonts
 - 1.3 Icons
 - 1.4 Misc
 - 1.5 Sprites
 - 1.6 Textures
- 2 Music Resources
- 3 Sound Resources

MORE PODDING ABOUT ...

Gamasutra has some interesting things to offer this month. For a start, there's a new GDC Radio podcast out about next-gen console development (http://www.gdcradio.net/2006/06/gdc_radio_presents_next_gen_co.html). Also available is a cool interview with the creator of Earthworm Jim, Doug Tennapel (http://www.gamasutra.com/features/20060606/murdey_01.shtml). He covers why fans are wrong and where EWJ on the PSP is going.

LEARNING FROM THE PROS

"I'm Jeff Tunnell and I've been 'making it big' in the game business for a LONG time. MBG lets you learn, for free, the lessons I paid a high price for." *Make It Big In Games* is a blog written by a professional indie game developer, offering insight, advice and instructions regarding many common problems that hobbyist developers face in their quest to become pros. Blog entries consist of topics such as "Five Realistic Steps to starting a Game Development Company" and "How Much Money Can Indie Games Make?", amongst many others. Visit <http://makeitbigingames.com/> to find out



4e5 BACK IN ACTION

The Gamedev.net Four Elements contest (<http://www.gamedev.net/community/contest/4e5>) recently released the list of element entries needed for its latest competition: Emotion, Economics, Emblem and Europe. The idea of the contest is to get entries centred around these four words, while still maintaining high standards of design, gameplay and technical expertise. The competition runs until the end of November and prizes include a free FastCapPro license for every contestant who enters. Time to get busy, developers!

Sound

3D Rendering

2D Drawing

Terrain Engine

Asset Repository

Input Handler

Game Classes

Scene Manager

Networking

Physics

ENGINES AND FRAMEWORKS

Most game developers understand that there are various game engines available that will make their development time easier. However a game engine by itself is often not enough. In fact it is often the case that a Game Framework is more important than a game engine.

A game engine is defined as "*the core software component of a video game. It typically handles rendering and other necessary technology, but might also handle additional tasks such as game AI, collision detection between game objects, etc. The most common element that a game engine provides is graphics rendering facilities (2D or 3D).*" (Wikipedia).

A framework is defined as a "*support structure in which another software project can be organized and developed. A framework may include support programs, code libraries, a scripting language, or other software to help develop and glue together the different components of a software project.*" (Wikipedia).

Based on these two definitions there is a very large intersection between a framework and an engine. A Game Engine is typically a very specific and clearly structured way of creating a

game. Often this is so rigid that the engine dictates what style or genre of game can be created using the engine. (for example ORTS is an engine for RTS games, Half Life 2 is an Engine for FPS games). On the other hand a Game Framework is only an outline, containing suggested methods and usually some code libraries etc. Game frameworks can typically be used to create any type of game, but often with a lot more work than with a Game Engine.

Game Maker can be considered a framework rather than an engine as it is designed to make any type of game rather than limiting the design options to a specific type of game. A simple example of this would be the various options available for user input. Game Maker has the facilities to allow the game developer to define how the user input will function. In an FPS engine these controls would typically be defined as part of the engine and will work as the player would expect for an FPS, while these controls can certainly be

extended it would be difficult to modify the user input to do RTS controls.

Many game developers consider building their own Game Engine before making their own games. These developers typically see the development of a game engine as a challenge to their technical abilities. As they develop their engine they are continually learning the functionalities and abilities of their chosen libraries. These developers typically spend years continually extending their engines with all the latest features.

Other game developers spend time learning to use the frameworks that are available to them. These frameworks are typically based on existing libraries or other game engines. These developers often are able to deliver completed games that make use of the features available in the framework. Often that means that these completed games do not have all the latest

DESIGN

functionality and hardware driven capabilities that are available.

Game Developers and Engine Developers are interdependent on each other. The greatest Game Engine in the world is worthless unless a Game Developer has been able to take the engine and create a game using it. For the Game Developer to do this the Engine Developer must extend the engine further than just being a library into a Game Framework that contains examples, code snippets and that defines the relevant structure games using the engine should use. At the same time Game Developers need to

embrace the game frameworks that are available as they contain the low level code needed to create top class games.

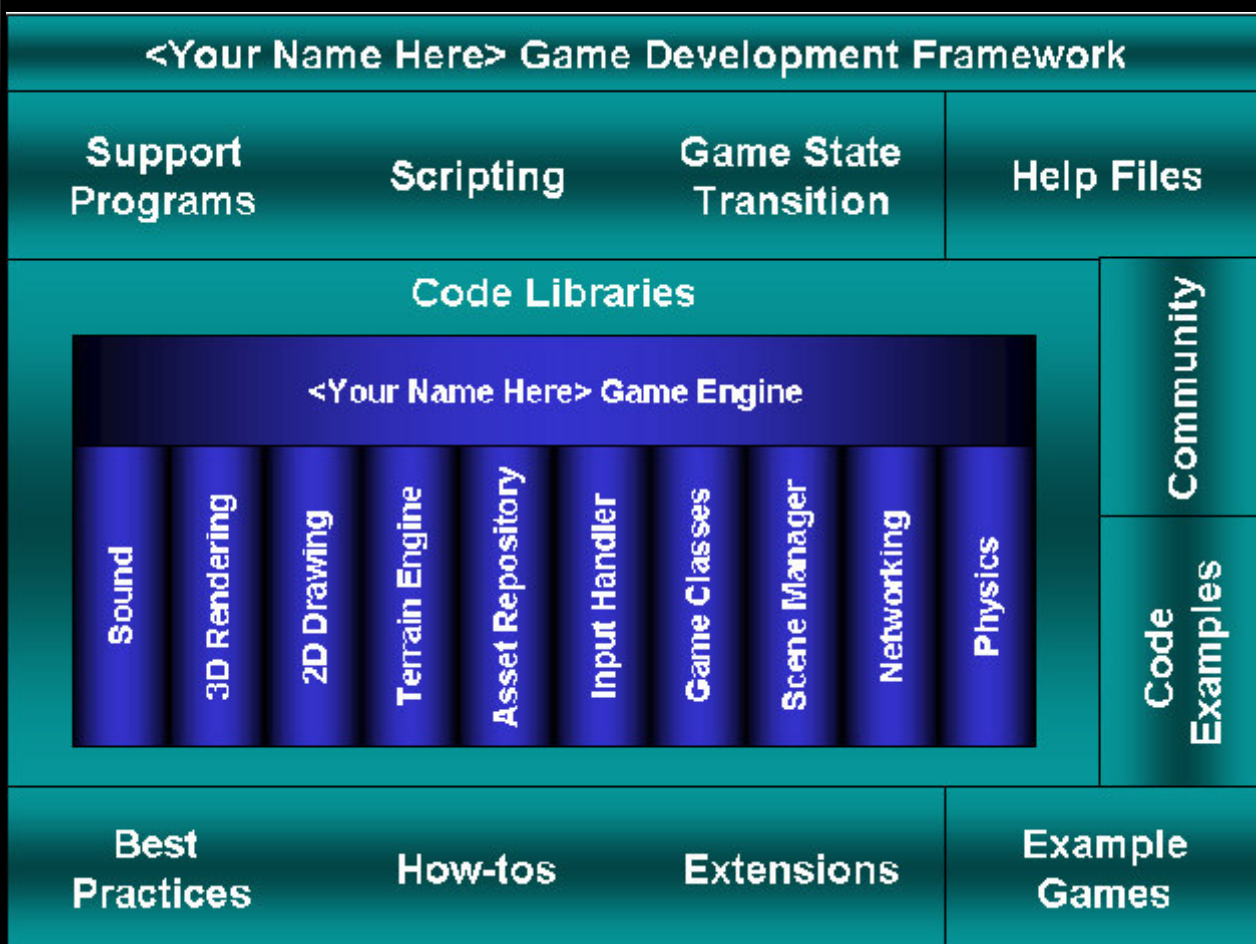
The best game developer using a poor game engine and framework will only be able to turn out a mediocre game.

The time has come for Game Developers and Engine Developers in South Africa to start communicating with each other. As a unified team we can turn out top class games that will rival what the rest of the world can produce.

CAIRNSWM

BY THE WAY ...

Game Maker is more than just a framework. Besides the Code Libraries, there are a number of tools to make development easier. These tools include an Image Editor, IDE, Compiler and Debugging. It also has extensive Help files that describe how to make use of the Framework, Libraries and Tools.



The above diagram outlines a typical game framework and the components that it consists of. The scope of a typical framework is often broader than that of a standard game engine.

Killer man eating worm From *Earth*

Start

High Scores

Help



An interview with Unc1354m (or Unclesam for the leet impaired), the creator of Killer Worm.

How long have you been developing games?

[Since] June\July 2005. I had always been interested in making stuff, and the Game.Dev section on the nag forums gave me the idea to download Game Maker 6.1. [I] spent a few days going through all the tutorials, [and] after about 4 days I released my first game.

Can you briefly explain the idea behind Killer Worm?

The idea in Killer Worm is that you are a giant, hungry worm. You need to eat, but pixel humans are so small [that] they don't offer much food, so you need to eat alot of them. Humans will start growing in technology and try to kill you since you are a huge threat to them; they grow from small pixels walking over the ground oblivious to the fact that there's a huge worm hungry under them, eventually having aircraft, tanks and the like in their name.

Killer Worm's concept is truly original. What inspired you?

The day Game.Dev's 08 compo was announced, I began brainstorming ideas instantly, [and] for some odd reason I just thought of an ant lion 'consuming' ants. It seemed a good idea, but instead [I] decided to make it a worm 'consuming' humans. [I] started work right away, ideas kept coming to me, [and] eventually I came up with the basic game dynamics.

What made you design the game in black and white, instead of colour?

This was laziness on my part. After making the sprite for the worm and pixel humans, I felt that they didn't need colour, but with a colour[ed] setting it looked messed up, so I decided to make the terrain and clouds black and white. About halfway through the game I got the idea to add old film scratches to blend the black and white in more. It came out pretty good.

Any current projects you would like to let us know about?

Outlaw Ball, an action platformer. This is a very large project, but [I] will keep the community updated.

Thread found here:

http://www.nag.co.za/e107_plugins/forum/forum_viewtopic.php?20717

Your game is one of the highest rated and downloaded games at the experimentalgame-play website. How do you feel about that?

I wasn't expecting it, since there are many other great games there, but as long as it stays at the view of potential downloaders I'm happy.

Do you see yourself expanding on Killer Worm at all?

In the future, maybe. It has been a success. If there's ever a time, [when] I need something to work on, it might be Killer Worm.

Is there anything else you would like to add?

Thanks to everyone who enjoyed Killer Worm or my previous games

DM

GAUNTLETS

Gauntlets is a top-down puzzle game. You control a tank and your aim is to progress through the different levels while setting high scores. The game features various upgrades such as speed, accuracy and attack that help you pass stages and defeat enemies and bosses.

The levels in Gauntlets are straightforward and don't really require much thought. All you have to do is simply time your moves and go when it looks safe. Of course, you also have to shoot the occasional enemy tank but they're not a focal point in the game. Control of your tank is accomplished through the use of your keyboard and mouse, where the keyboard controls the

The levels in Gauntlets are full of obstacles which you have to pass through using combinations of firepower, speed and accuracy.

movement of the tank and the mouse lets you aim the turret and fire.

Gauntlets also has its problems, though. One of them is its level design, the levels themselves being very repetitive (after completing the third stage, you'll know what to expect in future ones). The second problem with Gauntlets regards the camera control. The mouse lets you control the position of the camera relative to the field, which means you can move the cursor too far off the screen and lose sight of your

tank, allowing enemy turrets or tanks to shoot at you freely.

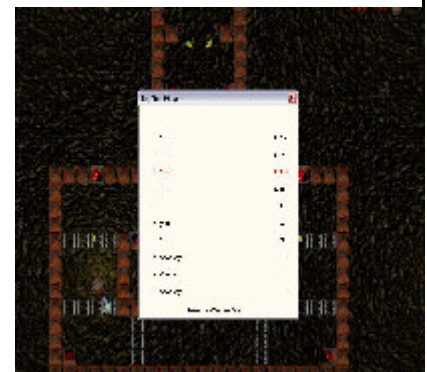
The last problem with Gauntlets is the respawn time. When your tank is destroyed, it takes less than one second to respawn, leaving you with virtually no time to prepare for any obstacles in the way or any enemies that were firing at you. A good example of this is the anti tank trap in the game. It is like a timed bear trap that opens and closes every few seconds. If you timed your passage over the trap badly and you die, you will respawn right on top of the trap and it's up to you to move your tank out of the way as quickly as you can.

Other than that, Gauntlets is a pretty fun game and if you take the time to play through a few levels, you may not be disappointed.

KNET



Gauntlets offers you upgrades that help you progress through the levels.



When you die, you set your high score and sadly, that's the end of it.

FIND IT!

http://www.apfstudios.com/gauntlets_beta4.zip

REVIEW



THE WORK AFTER THE WORK ...

I can't say I have been in the game development industry long enough to become a Jedi Master of the industry, but I have noticed certain trends that might help you. After almost 2 years of flogging my games about everywhere I can stick them, I stumbled upon a few gems that are other people's games. These people are just like me: beginners who love playing and developing games. Some were really innovative; others just filled up my hard drive. What I've noticed will definitely help me for future projects. At the level of development where you create games solely for the love of it and not for the want of any money, you get 3 types of games.

1. Teh SuXXor.

These games are not even half way decent, yet these abominations are thrown out to the world. Their creators want to show what they have, even though it is not really something special. Usually, these games are platformers with ripped sprites and very annoying midi music.

2. Middle-McDonalds

These games show some sign of innovation. They come from the minds that live to create something new, yet they fall off the bus somewhere in the middle, usually due to being riddled with bugs. These games, if done correctly, have the most potential.

3. The Cream

Now this is the interesting category. These games are spectacular for one reason: POLISH. Most of the time these game are not even innovative, yet the creators had a goal and stuck with their project, working countless hours to make it perfect. These games are the ones that have endless forum threads dedicated to them and that you bury in a special folder

on your hard drive.

Most of my own games fall into the second category. In fact, all of them do. What I have learned is that polish is king. A non-innovative game that took 4 months to make and was reworked to death is much better than an innovative game filled with countless bugs.

Why? When you are playing a game, you escape into the other world and every kink (bug) you find along the way snaps you back to reality. If you polish a game to every last bit, you create not only a game, but an experience. The longer you manage to suspend the player's disbelief, the better that player's experience will be.

I hate polishing my games and sifting through bugs. It sometimes feels as if the last bit, polishing, takes more time to do correctly than the entire game up to that point. However, once learned, the habit of polishing could be the best thing you have done to your game development life.

TROOJG

The popular Indie title, Eets, is a good example of a game which is simple at heart, but has been constructed with care.





ON THE BACK OF A NAPKIN:

PART 3: WHAT IS TEXTURE FILTERING?

Now that we've got a solid foundation to work with, we can start looking at more relevant issues in 3D. This week we'll explore the idea of texture filtering and why it's a good thing. We'll find out what the various types of filtering actually do, how they do them and how much of a frame rate hit each one causes. Read on if you want to know the difference between bilinear, trilinear and anisotropic filtering.

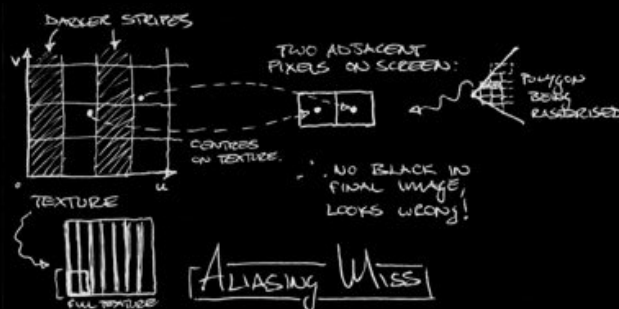
The reason that texture filtering is used in 3D is a small problem in the graphics industry called *aliasing*. Aliasing problems are very easy to spot and can ruin the visual illusion in a game.

Any computer screen is divided into pixels, duh. Each pixel can only be a single colour, it's impossible to have a pixel start off being red on one side and then fade to black on the other. The idea is that any image can be abstracted (see, it IS all lies, even your screen) by splitting it into enough individual pixels, unfortunately that doesn't always work very well: We notice "blockiness" on diagonal and nearly vertical/horizontal lines very easily.

Aliasing issues:

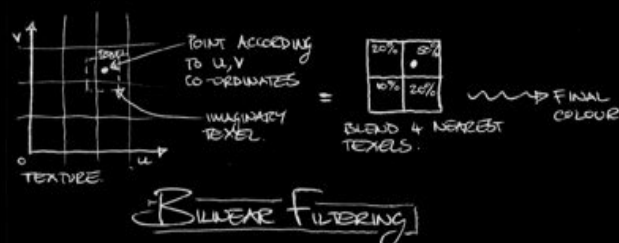
Aliasing is called that because it's the process of referring to one thing by a set of different handles or names. In this case, we're trying to get the pixels in a rendered image to refer to the pixels in a texture. As each pixel in our image is rasterised, (if you have no idea what that means, read the article on Vertices again) interpolation gives us a unique set of texture co-ordinates that tell us where on the texture to fetch the colour our pixel should be. That sounds complex, but it isn't: Rasteriser starts on a new pixel onscreen -> interpolation gives us the various values that pixel needs (by blending between the various vertices) -> texture co-ordinates give us an x and y point on our texture -> pixel is made that colour, with some adjustments for lighting and all that jazz.

Groovy. But textures are also made up of pixels (which we call *texels* to save on headaches), so they can have aliasing issues of their own... Damn. Here's a picture of some of the problems:

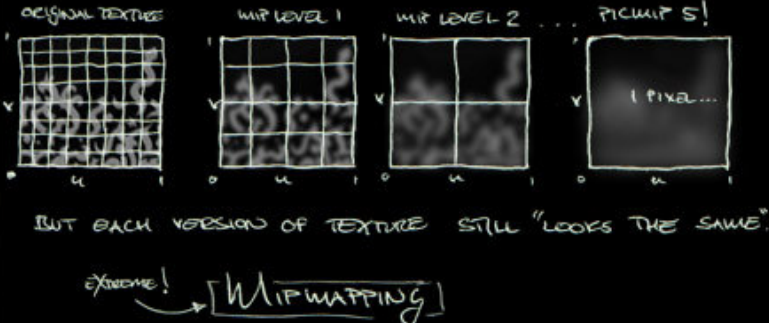


Bilinear filtering:

All you have to do to see the effects of bilinear filtering is to run almost any game in software mode and then again with hardware acceleration. Bilinear filtering makes textures "smoother" and less blocky by grabbing four texels near the sample point and averaging their values to get a blended colour for the screen pixel. It's this "blurring" that smooths out the textures on the screen and avoids aliasing misses.



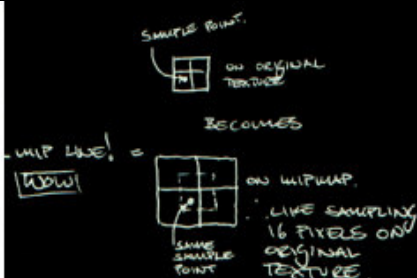
There are a few problems with bilinear filtering though. The first and most visible is caused by mipmapping. Mipmapping is a technique used to manually limit aliasing issues by providing smaller versions of textures that an engine uses when objects are far away, this means that there are less texels that it's possible to miss when there are large "gaps" between sample



points... Some engines use many levels of mipmaps, especially if it's possible to see very far into the distance.

Trivia: The famous "picmip 5" setting that Quake3 pros used simply scales down all the textures in the game, making a 512x512 texture effectively a 64x64 or 32x32 image instead. This blurs all the textures like crazy, but that's not why the pros did it: They were after the small increase in FPS caused by having smaller textures and less texel lookups and a rather debatable "visibility increase"... Oh what crap textures you have grandma! All the better to see you with dear.

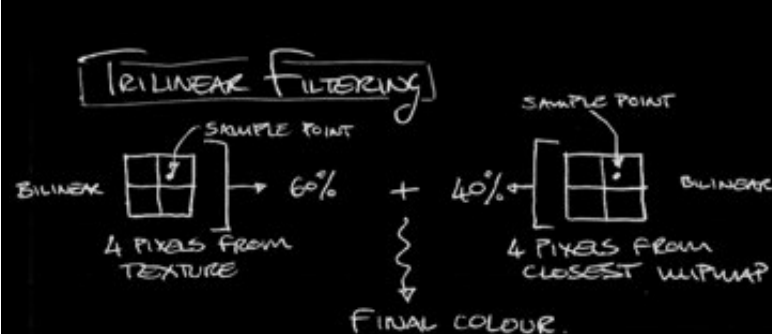
So, mipmapping was invented before bilinear filtering as a way to deal with distance aliasing issues. The smaller textures (remember how the U and V texture coordinates only range from 0 to 1? The different sizes of mipmapped textures are one of the reasons for that) allow for less "misses" of texels because there are less texels in total. But, when you're using bilinear filtering AND mipmapping, the smaller textures are blurred a lot more by the bilinear filter:



This sudden increase in blur is what we see in games as a horizontal or vertical "line" on floors and walls, especially when moving. It ends up looking like there's an error that stays a certain distance ahead of us in the game, which can get very frustrating. That's why there's the option to turn on trilinear filtering.

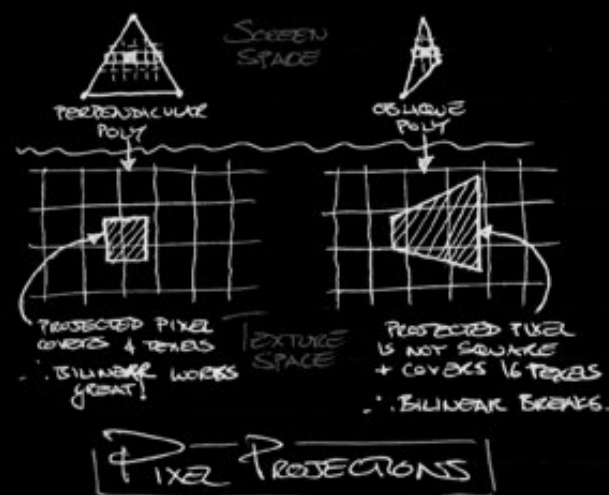
Trilinear filtering:

Just as *bilinear* filters across two dimensions, *trilinear* filters across three. Except that the third dimension is the Dimension of MipMapping! This means that where bilinear filtering grabs four texels and averages them out according to an algorithm, trilinear grabs eight texels (four from one mipmap and four from the other) and again averages them out to get a final colour for the pixel on screen.



Anisotropic filtering, the next level:

So, both bilinear and trilinear filtering work in texture space to try to calculate the correct colour for a textured pixel. Unfortunately this isn't always the best approach: It works fine when the textures that are being filtered are displayed on polygons that are at right-angles to the camera, but it's a poor approximation for polygons that are at non-perpendicular angles. This is because of the shape of a pixel on screen when projected into texture space depends on the angle of the polygon the texture is being used on. Wait, that sounds confusing... Here's a picture:



Anisotropic filtering takes this difference in mapping into account and uses many samples of the texture in patterns that depend on the projection to calculate the final colour of our on-screen pixel. Unfortunately ATI and NVIDIA use different patterns and sometimes even different amounts of samples to arrive at their final values, so it's not really possible to draw a simple snapshot of anisotropic filtering. It is possible to mention that anisotropic filtering uses a lot more texture samples per pixel (obviously) so both card manufacturers decided to allow us to have some say in the amount of bandwidth vs the visual quality of anisotropic filtering by giving us the arbyly named 2x, 4x, 8x and even 16x Aniso settings that we can tweak in our drivers.

But what does that all mean?

Why don't we take a step back and figure out what all this means for our gaming?

Texture filtering makes games look better by making our textures less dependant on resolution. Of course, we could simply up our resolutions and make our games look smoother and crisper that way. That's option 1, but it does mean that our whole rendering pipeline is calculating a lot more pixels, so your FPS will depend on the speed of your GPU's core clock.

If you can stand it, turning filtering off (and living with only point sampling and mipmaps) is the fastest approach in terms of memory bandwidth. It doesn't look great at all though...

Bi- and tri-linear filtering are the current standards because an average graphics card these days has a memory clock that's fast enough to allow 4 (for bilinear) and 8 (for trilinear) texture-memory reads per pixel. So, depending on your card, you can probably afford to use either of those filtering methods without taking a FPS hit at all.

Anisotropic filtering ups the memory reads per pixel quite dramatically, sometimes even doing as many as 128 reads on the highest settings! So your memory clock speed is really important if you want to use aniso. Newer cards can handle the lower levels of aniso (4x and lower) without pushing themselves too much, both manufacturers use optimization

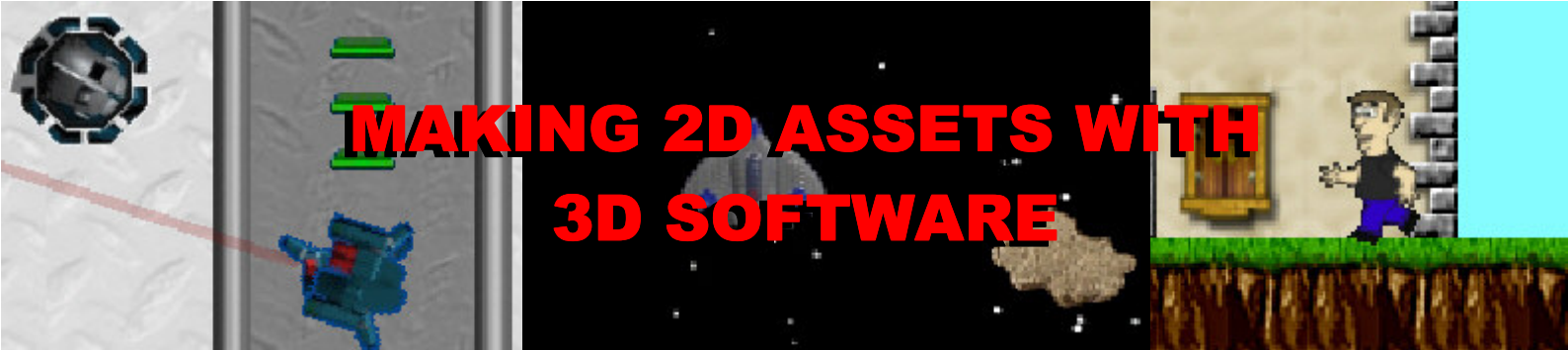
algorithms to attempt to apply anisotropic filtering to only those parts of a scene that need it.

If you really hate fuzzy textures, enable the highest level of aniso filtering you can, but it will give you slower fill rates as each pixel reads tons of memory. The slowdown will get worse the larger your resolution, but such a high level of filtering might make it tolerable to take your screen size down a notch or two. But that's all purely personal choice.

The bottom line:

- Point sampling = simple + fast + blegh.
- Bilinear = "blurs" textures, effective but has nasty mip-lines.
- Trilinear = fancy sounding slower upgrade to bilinear, kills mip-lines and not much else.
- Anisotropic = different method (actually uses bilinear as samples), tries to match screen-space by doing more work, slowest + prettiest.

DISLEKCIA



MAKING 2D ASSETS WITH 3D SOFTWARE

What are the most wanted people in the Indie game development world? Well, I think you may know already. Artists. Having a graphics artist in your group is like having superpowers. Yet every one of us must have the creative gene in us, otherwise we wouldn't be making games. What makes it so impossible for us to make our own assets?

These days, the artists out there concentrate on 3D modelling, as they should. It is where the industry is going. But we, the Indie game developers, need 2d sprites or animated .gif files to do our job. So how can we still use the 3d models produced by today's artists in our games? Well, that's exactly what I'm trying to show you all here in this series of articles. We will run through the various steps I follow to make working assets for my games. Now, I'm not an expert, not nearly, but I can handle a 3d modelling package. This tutorial should be universal to all programs out there – the package used must just be able to “render” a scene. So, let's get started.

For these articles I will be using 3D Studio Max (any version will work), Paint Shop Pro 7, and Game Maker 6.1 (registered).

Setting up the scene

This, if any, is the *most important* step in the process, and if not done correctly will cause a lot of work to pop up later. In your modeller import the model you would like to use in your game. Most cases you would like to get a top view of your model, or a side view. For my “golden lamp” I will be using the top viewport as my rendering viewport. The use of cameras are advised for Isometric assets, as you would like to setup up a template scene first, that is used in all the assets, to keep informality for post editing and style.

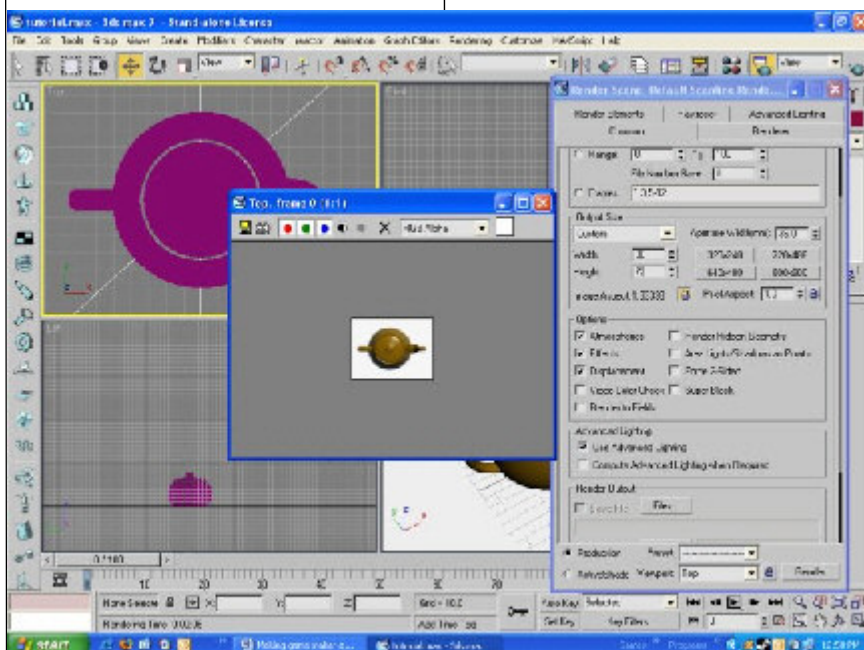
You should also use a plane in the same color of your games environment. For example, if it's a space shooter a black plane may be needed, or a desert shooter an orange plane would be good. This will help loads in the editing of the “near opaque” area of your images. For my lamp, I am going for a white plane, to make it more universal. So, I create my lamp and place it in the centre of the field (0, 0, 0). I then use the Max function to extend the view to fit all objects (most programs have this function). I add a huge plane at co-ordinates (0, 0, 0) and make sure it does not overlap the main model. When moving and adding objects, I do not use the top view port which I would later use for rendering, as one can easily resize or move the viewport without knowing, and that is not desirable. Now you should apply the UV Map and texture you would like on the object, and also the required material settings you want to use.

For this situation I created a Raytrace material with metallic settings, and a gradient reflection setting. My diffuse map is a nice goldish color.

Next we need to pre-setup the renderer. We need to be sure on the size we require. I usually make the renders slightly bigger than the size I need in game maker. The size I chose is 96x72. This added resolution helps loads when post editing. I adjust some custom settings and I do my first render.

This is really simple, but usable. Next month we will add lighting to the scene and animate the model. In the mean time, you are all welcome to collect some free 3D models from <http://www.3dtotal.com> and <http://www.3dcafe.com> for the next issue, and we can make some nice assets. See you next month.

HIMMLER





PART 1: MINIMUM REQUIREMENTS

A complete game is more than just gameplay. A complete game must be equipped with easy access to features, information about the game and its creators and options to adjust the experience of play. In this article, CAIRNSWM and FENGOL cover some of the basics that a game needs to “feel” complete.

These features are mostly slog work that creative and most hobbyist developers leave out due to lack of enthusiasm or time constraints. However, players notice the lack of attention to these details and they're worth spending a little extra time on.

Splash Screen: A good splash screen serves two purposes in a game. Firstly, it acts as an immediate indication to the player that the game has started, while the game loads in the background. Secondly, it gives a new player an indication of the theme of the game.

Main Menu: A clearly defined main menu gives a game a very nice completed feeling. The Main Menu must be clear, responsive and rather intuitive, and should make it easy for the player to access the various other game features or exit the game without using Alt-F4.

Tutorial or Help Screen: More often than not, a new player will download your game and dive into the action without exactly knowing what to do; only if he can't figure it out for himself or is struggling to get very far with the game will he look for a help or tutorial button.

If a player can't work out how to play your game within a few minutes, the chances are good that they will just delete your game and forget about it. Rarely does a player think about or spend time looking for a readme file in the project directory, so clear and easily read tutorial or help screens will keep the player from getting frustrated.

Credits: Hobbyist developers rarely create their own content for their games; we rely on graphic artists, midi musicians and other artists for content, often for free! A credit screen is a good place to thank the people who've helped make the game what it is.

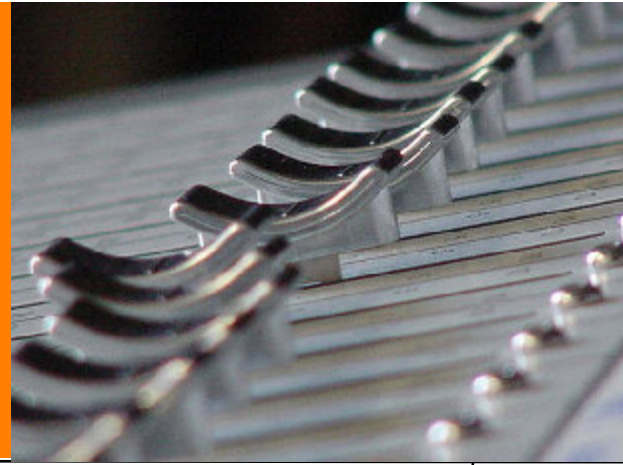
About Screen: While this could easily be part of the Credits screen, the About screen serves a different purpose. The About screen gives the game's version number, copyright information, the game's homepage where the player can get more help or information and an email or link to provide feedback. The About screen can also give more information about you, the hobbyist developer; your email address, website or blog, and other games you've made.

Readme File: While a player won't necessarily read the readme, it's important to include a readme for logistical reasons. The readme should include: licensing information (may someone sell, give out or distribute your game without your permission?), the various copyright holders' information (in fact, include all the information from the Credits and About Screens), a list of known bugs and issues (if there are any that you are aware of) with work-arounds and a history of what's changed in your version updates.

These features make it easy for the player to get involved in your game and it's worth spending time adding them to your development cycle. Practice by always adding these features, and they won't be the hard work that they may appear to be at first.

In the next article, we'll cover “adjusting the user experience” where players should be able to control aspects of your game like music volume, graphical effects and speed performance.

Cairnswm, Fengol



THE BASICS OF SOUND DEVELOPMENT

In the vast world of audio, our jobs become more and more complex as the ever-evolving matrix of technology pushes us into having to be multi-skilled professionals, where a single person can now do the job that an entire team of skilled individuals did in the past. As our computers get faster, our software becomes more and more complicated, allowing us to live in a world where the boundaries of our capabilities are set only by the limits of our imagination. It is for this reason that I would like to start off this audio column by introducing to you some of the "tools of the trade" used in the audio industry today. But with such a vast array of powerful tools at our disposal, some of us can't help but feel a little bit overwhelmed at all these technical gimmicks that promise to make our games sound like they've just stepped out of Mr. Lucas' sky-walker ranch.

Sure, some software might perform certain tasks better than others, some are designed only to do one certain thing, and some might even claim to do it all! But the question is: Which one is the right one *for you*? Let's talk more functionality and less gibberish. Let's help you spend your money wisely on a product that best suits your needs for making successful game audio.

The first question that you would naturally have to ask yourself is: "How

much am I willing to spend?" There are plenty of choices that will suit any budget, whether you want to spend as little money as possible or if you're willing to give an arm and a leg for it. Luckily, there is something for everyone, even if you're not prepared to spend anything at all!

The External/Hardware Sound Card:

(Assuming that you already have a computer) The sound card is the piece of hardware that acts as the "audio brain" of your setup. It's an interface which allows you to connect external sound sources, and be ready for recording. Most sound cards feature multiple inputs, such as microphone inputs, line inputs (keyboard/guitar etc) and, usually a MIDI In/Out/Thru feature will also be available (amongst others). If you want to take sound seriously, one of these babies will definitely need to be by your side. It's a fair investment that will reap many audio advantages.

The "Multi-track" software sequencer:

These are generally the most popular software tools. These "all rounder" tools, and cater for most audio tasks such as general editing, music creation/

sequencing, audio/visual editing, recording and even post production work. Expect to pay a fair amount for this kind of software, however, as they hold seemingly endless features. Also be sure to have a somewhat powerful PC at your side with at least 1 GB of RAM; these tools are normally quite process intensive. External equipment is available to take some of the load off the CPU, but we'll leave that topic for a future article.

A free sequencer to check out is Kristal Audio. (www.kreatives.org/kristal) This type of software is generally hard to come by as a free tool due to the complex tasks that are required from it. Kristal has quite a few limitations, but is a good introduction to sequencers nonetheless.

The Audio Editor:

This is the tool which I feel is most relevant to us in the small game development community. This is not only because these tools are quite simple and cheap, but because the type of functions that this kind of software performs very much apply to what we are doing.

The editor is a sound designer's best friend. With excellent visual

Some other popular sequencers:

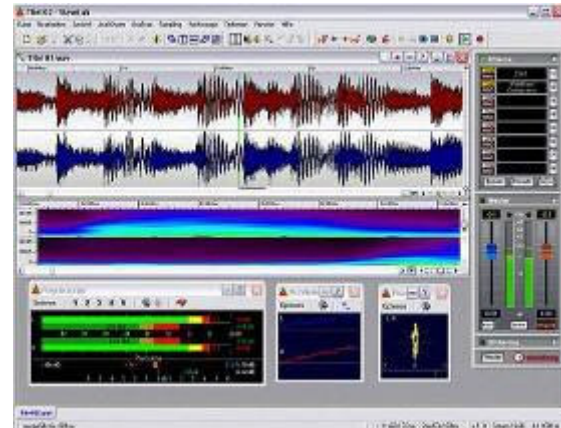
Steinberg's "Cubase SX"
(www.steinberg.net)

Apple's "Logic"(MAC)
(www.apple.com/logicpro)

Digidesign's "Pro-Tools"
(www.digidesign.com)



Left:
Digidesigns Pro-Tools LE 7. The industry standard.



Right:
Steinberg's Wavelab 6. A common favourite amongst audio junkies

interpretation of your audio wave, you can literally "read" your sound like a book allowing you to decipher what the next critical audio decision will have to be. And if that isn't enough, you'll have an entire militia of different types of metering tools at your side which allow you to critically analyze your wave. The main features that we will be focusing on with the audio editor will be:

- *Monitoring our mix, (checking for clipping/distortion)*
- *Checking overall levels, (consistency in sound, correct compression applied)*
- *Multi-track capabilities, (the layering of our sound effects and overall mix)*
- *EQ and overall effects applications, (plug-ins)*
- *Proper conversion/compression of our sound files, (eg. WAV to MP3 conversion)*
- *The "batch converter", a common feature amongst audio editors that allows us to convert multiple audio files at once! (More detail in the near future)*

Two free(ish) editors really worth checking out are:

Goldwave editor

<http://www.goldwave.com/>

This one can be continuously kept in demo mode (and is very affordable none-the-less)

Audacity

<http://audacity.sourceforge.net/>

This one is 100% free and is really quite good! I would recommend Audacity as really good starting-point/budget friendly audio editor.

Audio Plug-ins:

Plug-ins are stand-alone software applications that work independently, but still use your sequencer/editor as a host. These become very useful when you require something specific that your editor might not cater for. Just open it through your VSTi/DXi link and, presto! Your editor has become that much more powerful.

There are literally thousands of plug-ins available, some costing up to R90000! Yes, there are many good free ones too, but for the sake of trying to keep this an introductory column, I would rather go into more detail on plug-ins at a later stage.

Sound-Effects Libraries:

Your sound-effects library is the source of all your sounds. These too, come in all shapes and sizes. Available are general "all-rounder" libraries with everything in them, from the sound of gun-shot effects, planes, right through to

someone walking on wet lawn during a windy day, or more specific categorized libraries.

A really good place to get free samples is the free-sound project, an online sound sharing community that I can highly recommend.

<http://freesound.iua.upf.edu>

Also be sure to check out

<http://www.sound-ideas.com> (a popular source for the professional industry-features an excellent gaming library too!) and

<http://www.hollywoodedge.com> (another popular choice for the professional industry).

Remember, whatever choice you make for your everyday audio tool, make sure that the software you use works for you and doesn't block your creative process. A tool is there to enhance your creativity, not to make your life complicated. As time goes by and you try more and more software, eventually you will found the right one -- the one that works best for *you*!

As the months go by, I will go into more and more detail about these topics. I believe that, to get into actual production, we need to become familiar with our tools and understand the basic fundamentals of audio.

ZPHYR

Other popular editors:

Steinberg's "Wavelab"

(<http://www.steinberg.net>)

Sony's "Sound-Forge"

(<http://www.sonymediasoftware.com/Products/ShowProduct.asp?PID=961>)



RECURSION

Recursion is yet another one of those fancy terms that computer scientists use to describe a technique that is actually quite simple in practice.

At its base level, it is merely a method where a function does some work, then “calls” itself to do the same work again. However, instead of using the data that was used at the start of the previous work, it uses the data that was created as a result of the work. This process is repeated until a certain condition is met, at which point the recursion ends by returning from the current level of the function, instead of continuing into another recursive call of itself.

Recursion is typically used to apply what is known as the **divide-and-conquer** approach (mentioned briefly in last month’s article), which is made up of the following three steps:

Divide:

Divide a problem into a smaller number of “sub-problems”.

Conquer:

Process each sub-problem by using recursion until the “size” of the problem is small enough to be solved in a straightforward manner

Combine:

Combine the solutions for each sub-problem to create a solution for the original problem.

A typical example of recursion being used is the **merge sort**. Merge sorting works by recursively breaking up a list of data into two smaller lists, each of (**list_size / 2**) size, until there is only one element in the list at the current level of recursion. At this point, the recursion is ended and the function returns

to its previous level where it merges the data of the two lists into one:

```
MergeSort(data, low, high)
{
    if(low < high)
    {
        mid = (low + high) / 2;
        MergeSort(data, low, mid);
        MergeSort(data, mid + 1, high);
        MergeData(data, low, mid, high);
    }
}
```

It is quite easy to see where the recursion happens in the above pseudo-code. **MergeSort** calls itself with modified data until **low** is the same value as **high**. At this point the recursion stops, the function returns to its previous level, and the **MergeData** process begins on the list of data. Once the data is merged, the function returns to its previous level and the merge process continues until the original level of the function is reached, at which point the data has now been sorted completely.

Now that you know what recursion is, it doesn’t mean that you should run out and make use of it all over the place. The real trick is to know what problems should be solved by implementing a recursive algorithm, and what problems should instead be solved with an iterative approach.

COOLHAND

MOBILE GAME DEVELOPMENT IN JAVA



PART 2: MAKING MOVES

Welcome to the second tutorial in cellphone game development. Let's start off by reviewing the MIDlet from the previous tutorial. The first thing to notice is our two class declarations: the first being our MIDlet class, which is the mobile application that the system runs. The second is the Canvas class that is used to actually display objects. As you can see, our MIDlet doesn't do very much other than create the canvas and set it as the MIDlet's main display. The three methods, *pauseApp*, *startApp* and *destroyApp*, are called respectively when the MIDlet enters the pause state, the running state and when it is destroyed. It is important to note that *startApp* may be called more than once, so what we've done by creating our canvas in this method is technically incorrect. Within the TutorialCanvas class we have overridden only the *paint* method. As you can guess this is called

when the Canvas is repainted. The actual paint implementation simply clears the screen to white by setting the colour, then drawing a rectangle the size of the screen. We then set the colour again (to red) and draw some text.

To extend this simple application to do more than draw text once, we need an update loop. As can be seen in [Listing 3](#), this is achieved by implementing the *Runnable* interface in our Canvas subclass. We also add a static Thread member variable, *gameThread*, and a boolean member variable, *exit*, to allow the thread to be stopped. The new constructor simply assigns a new Thread to *gameThread*. The *run* method is where we do our work. For now we just check if we need to exit, and if not we request a repaint (and wait for that repaint to happen) and pass

control to another thread. There are two important things to note here: firstly, never explicitly call *paint*; use the safer *repaint* instead. Secondly, the only way to stop a thread in J2ME is for it's *run* method to finish. The programmer is required to ensure that all threads they start are stopped when the MIDlet is destroyed as the JVM is not guaranteed to do this automatically.

This brings us to the changes to the MIDlet ([Listing 4](#)): We have made the canvas a member variable, set it as the display, and ensured that it's initialised in *startApp*. In *destroyApp* we make sure the canvas's thread will stop. We have the beginnings of a game loop, so let's add some game-like behaviour. We need a ball in our game, so let's add that to our Canvas and give it a simple bouncing animation. There is no support for floating point numbers in

Listing 1. Class declarations.

```
public class TutorialMIDlet extends MIDlet
..
class TutorialCanvas extends Canvas
```

Listing 2. Paint method implementation.

```
g.setColor(0xffffffff);
g.fillRect(0, 0, getWidth(), getHeight());
g.setColor(0xff0000);
g.drawString("Hello World", 0, 0, 0);
```

Listing 3. TutorialCanvas updated to implement Runnable.

```
class TutorialCanvas extends Canvas implements Runnable
{
    public static Thread gameThread;
    public static boolean exit = false;

    //Constructor
    public TutorialCanvas()
    {
        gameThread = new Thread(this);
    }

    //run the game loop
    public void run()
    {
        while(!exit)
        {
            //paint everything
            repaint();
            serviceRepaints();
            //give other threads a chance
            Thread.yield();
        }
    }
}
```

MOBILE

the J2ME configuration we're using, (CLDC1.0) so we have to settle for two integer values for the ball's position. We also add velocity values for the ball, and define a constant for the ball's size. In our constructor we set some initial values for the position and velocity. In our update method, before we call *repaint*, we update the ball's position. This is easy enough, since we just have to add our velocity and change direction if we are outside the bounds of the screen. We actually allow the ball to slightly exit the screen because it's a bit simpler this way. Finally, we replace our text with the ball in the paint method. We use the *Graphics.fillArc* method to specify our circle. Note that the position in this method refers to the top left corner of the rectangle that would contain the arc, not the origin of the arc. All of these changes can be seen in

Listing 5, and, as always, I have omitted code that has not changed.

That's all for this lesson. We now have an update loop running in a separate thread that is animating a bouncing ball. The code as it stands has a lot of room

for improvement and optimisation, but my focus has been clarity, so I'll leave all that as an exercise for you. Next time, we'll delve into the MIDP game engine.

FLINT

Listing 4. Updated MIDlet class.

```
public class TutorialMIDlet extends MIDlet
{
    TutorialCanvas canvas = new TutorialCanvas();

    //Constructor
    public TutorialMIDlet() {}

    //Called when the app starts
    public void startApp()
    {
        Display.getDisplay(this).setCurrent(canvas);
        TutorialCanvas.gameThread.run();
    }

    //called when the app is destroyed
    public void destroyApp(boolean unconditional)
    {
        //make sure we stop the game thread
        TutorialCanvas.exit = true;
    }
}
```



A ball bouncing around the screen.

Listing 5. TutorialCanvas updated to include a bouncing ball.

```
..
static int ballX, ballY, ballVX, ballVY;
static final int BALL_RADIUS = 5;

//Constructor
public TutorialCanvas()
{
    ..
    ballX = ballY = ballVX = ballVY = 1;
}

//run the game loop
public void run()
{
    ..
    //update our ball position
    if(ballX < 0 || ballX > getWidth())
    {
        ballVX = -ballVX;
    }
    ballX += ballVX;
    if(ballY < 0 || ballY > getHeight())
    {
        ballVY = -ballVY;
    }
    ballY += ballVY;
    ..
}

//paint the canvas
protected void paint(Graphics g)
{
    ..
    g.setColor(0xff0000);
    g.fillArc(ballX-BALL_RADIUS, ballY-BALL_RADIUS,
              2*BALL_RADIUS, 2*BALL_RADIUS, 0, 360);
}
}
```

THE TRUTH ABOUT INSTITUTIONS

So, off to a tertiary institution? Studying for your future? Excited about the opportunities in professional game-making that your studies will present? Well, know now that there's another side to going to university, something that's overlooked by a lot of students. You're going to be surrounded by tons of interesting people while you're studying, and if you don't use that time to make contacts and learn about things "not in your field", you're missing a big opportunity to get ahead as a game creator.

How does one go about learning, though? Well, one of the best ways to get to know different people is to take elective courses that are totally off your "stream". If you're careful and study your university handbooks, you'll be able to get credit for them no matter how strange they might be. A good way to decide if something is as interesting as it sounds is to spend a bit of time exploring exotic-sounding classes in first year, once you're settled in it's really easy to just sit in on a couple of lectures. If the course is interesting, relevant to you or has types people that you know you're going to need to involve yourself with during your working life, sign up for it next year.

Here are a couple of ideas for things that you might be able to use to strengthen your CV and network at the same time. You should always keep a hard-copy of your contacts and how / why you know this person and what they can do.

ART / DESIGN COURSES

If you want to make games, you're going to need art resources. Making friends with the arty students can lead to some of them wanting to help you with your games and providing graphics that you would never be able to do.

You don't have to do hectic art classes either, most universities offer "lighter" courses such as Visual Communication or Semiotics. While they can be very strange to a technically-minded BSc student, learning the theory behind perception or how colours define emotions can be very useful and immediately applicable in your games. At the very least, you'll have some understanding of how to talk to artists in their own language afterwards – never underestimate the importance of good communication!

I've got a short-list of contacts from my design classes that I send work to every once in a while when people ask me for business cards or logos. They're all keen to work with me on games and I send them playable versions of what I'm messing with every once in a while. One of them even likes GM because she can change the graphics without having to code.

The same goes for music classes, but typically those are much harder for "lay-listeners" to understand. I'd suggest getting to know some of the design students and hanging around in the "art" areas of campus for a while. You'll meet the musicians eventually...

BUSINESS / MARKETING COURSES

If you're planning to do it alone and start up a company to build games, knowing all the little things you need to do to incorporate and understand how investment works can be invaluable.

You should be able to find entrepreneurship courses without too much difficulty and they'll probably fit into your course-structure quite well. Once again, they'll be a bit simpler than your tech-minded CS courses, but it's a different method of thinking. Once you've got an understanding of business, you'll know if it's something you can do or if you'll need to get a partner to run the business side of things. The people in these courses can be valuable if you have questions later when you start up.

The dark art of marketing is a driving force behind the success or failure of games these days. Getting into a couple of marketing courses will give you useful contacts if you need marketing help, plus you can continue with it through to second-year level to give you an edge – if you're built to withstand the lovely task of manipulating people all day, of course!

ENGLISH COURSES

Not only are some English modules excellent for helping you develop that story-writing side of yourself, there's also a whole lot of information about the publishing industry that can be learned here. Yes, the basic principles of publishing are the same for books, music and games, so a good publishing course will equip you to deal with publishers on a much better footing.

Personally, I'd recommend getting to know the publishing /



English students through the first-year art modules you take, they'll probably be in those. Then once they start talking about things that interest you, start sitting in on the odd lecture.

PSYCHOLOGY COURSES

A little bit of psych can be really useful in helping you decide how to invoke powerful emotions with your games. Unfortunately most 1st year psych modules are very low level, but if you feel that you're learning something applicable, go for it.

Usability and Human-Computer-Interaction courses: Most informatics faculties will have a few courses dedicated to these fields. In the gaming sphere, the most important person you're ever going to interact with is your user – the person playing your game. Any techniques and skills you can learn to make their experience better will translate into success later, even if you're simply a programmer...

While these courses are most useful for aspiring game designers, learning the ins and outs of usability will make your code better and less error-prone, despite what the hardcore CompSci guys say about it. Trust me. Being able to make a piece of software fun and understanding WHY it's fun will serve you in good stead.

REGULAR COURSES

Most CS degrees will have you doing at least some maths. Go to it and work hard, you can do ok without it, but knowing your algebra makes graphics coding much, much easier. Trying to understand AI without combinations, permutations and some calculus will break your head.

Don't run away from the physics course. It's not as hard as you think, provided you keep working on it every week... It'll help you out in the days of physics cards and emergent gameplay.

There are tons of other courses which might prove useful. Remember that you need to sit down and decide what you should focus on and what not. Your degree will try to push you towards certain things, but remember that this is all for you to use – get the most out of everything you can.

DISLEKCIA



www.itintellect.com
www.itintellect.com

Durban
Ph 031 277 2000
info.dbn@itintellect.com

Bryanston
Ph 086 1 484 484
info.gp@itintellect.com

Cape Town
Ph 021 421 8555
info.ct@itintellect.com

Richards Bay
Ph 035 789 3115
info.rb@itintellect.com

Pietermaritzburg
Ph 033 386 6057
info.pmb@itintellect.com

Bloemfontein
Ph 051 447 3635
info.bfm@itintellect.com

DON'T PLAY GAMES...

B A: DEGREE IN GAME DESIGN

B S C: DEGREE IN GAME PROGRAMMING

N C I T: NATIONAL CERTIFICATE IN INFORMATION TECHNOLOGY

i.t. intellect
COMPUTER **GAMING** SOLUTIONS



O N L I N E

devmag.googlepages.com