Adonthell Graphics Documentation v0.1

James Nash

31.08.2002

The Adonthell Graphics Documentation

The AGD is maintained by:

Benjamin Walther-Franks <bwf@tzi.de>
James Nash <cirrus@linuxgames.com>

If you have found any typos or have suggestions, comments and/or new tips please mail them to one of us - cheers!

This document is part of the Adonthell Designer's Documentation
http://adonthell.linuxgames.com

Created with LaTeX2$_\varepsilon$

Contents

1 Intro

1.1 What is the AGD?

These documents are intended for artists wanting to create graphics for Adonthell or games based on the Adonthell engine. This is where you find out how to get your pictures from your graphics program into the game.

The AGD does NOT cover how the graphical parts of the engine are programmed. For details on this please refer to the programming documentation.

1.2 Overview

There are several more or less independent types of graphics for games using the Adonthell engine: Window gfx (window borders, buttons, fonts etc..), In-game gfx (map-gfx, character-gfx, items etc..) and Cut-scene gfx (those fancy animation sequences). Thus the layout has been designed so that you can quickly find all the information relevant to making the gfx you are interested in. Essentially, just read the (sub-)chapters that cover the aspect you are interested in and ignore everything else. The only exception is that all artists must be familiar with the conventions outlined in chapter 2 and if you encounter chapters called 'General info on ¡somethingorother¿' you should read them too before moving on.

You can use any graphics program to do your work but you will need to use Adonthell's developent utilities to convert your graphics files into something the game can understand. As these are not normally included with the binary releases of Adonthell and frequently get updated we recommend you use the sourcecode from our CVS (http://savannah.gnu.org) and compile yourself.

Occasionally you will find sections marked 'Adonthell specific'. These cover rules and conventions that only need to be followed if creating an Adonthell game. If you are working on a game that just uses the Adonthell engine you can ignore these.

Furthermore you will encounter boxes marked 'Tip'. These contain tips and tricks that our artists have discovered while making gfx for Adonthell. You don't *need* to read them if you are in a hurry but they are well worth a look.

2 General Info on the Gfx Conventions

We recommend that all artists read this chapter as it applies to all areas of creating graphics.

2.1 Gfx format

The Adonthell engine has its own image format (y'know, like .jpg or .tif) to store gfx. Essentially they are gzipped 16-bit bitmaps. For us artists the relevant part is that they are 16-bit - that means 65,536 colours. It may not be much compared the the usual 16.7 million you get with 24-bit colour but it's more than enough for Adonthell's purposes and it means smaller filesizes which means better performance and smaller downloads!

Now then, as you may or may not know most common graphics applications (Photoshop, GIMP and Co.) will not let you save as 16-bit images. Fear not though - Adonthell comes with a conversion utility that converts 24-bit raw PNMs into the game's own 16-bit format. This means you save your work as a 24-bit raw PNM file and use the conversion utility. The converted file can then be used directly by the game!

Graphic files for the Adonthell engine may be any size you want. However, bear in mind that the game's resolution is fixed at 640x480. So, if you are making a house for example that has to be entirely visible on the screen you would need to make it smaller than 640x480. Also, smaller gfx will mean smaller files and thus load faster in the game!

Adonthell specific

For the Adonthell games we have decided that humanoid characters must not be larger than 40x80 pixels. This is so that buildings can be designed with doors measuring at least 40x80 pixels and all characters will be able to pass through

2.2 Transparency

With Adonthell you must differentiate between transparency and transluceny. When we talk of things being transparent we mean that they are completely see-through (ie: 0% opacity). By translucency we mean that something is not quite see-through (ie: any opacity from 0% - 100%). In fact translucency is just another term for the alpha level (as it is referred to in some gfx applications) or the opposite of opacity (which Photoshop and GIMP users will be familiar with). Opacity refers to how solid something is and transluceny refers to how see-though it is. We have to tell them apart as Adonthell's gfx format has support for transparency on a per pixel basis (like GIFs do) but NOT translucency. However, the engine can display entire images at arbritary translucency levels in the game (See fig. 1 below). Examples are the shadow beneath a character - the shadow is actually a seperate image from the character and is just a solid black shape but the engine is making it transparent as it displays it.

For now we will only concentrate on transparency. Graphics can only be stored as rectangular files, so if you want to 'cut out' a shape within an image you must make all other pixels transparent. To do this you ust make these pixles exactly magenta (that's Red: 255, Green: 0 and Blue: 255, or FF00FF in hex notation) and turn masking on in the conversion utility. Therefore images with transparency cannot have any visible true magenta pixels (you'll have to settle for 255,0,254 or something like that).

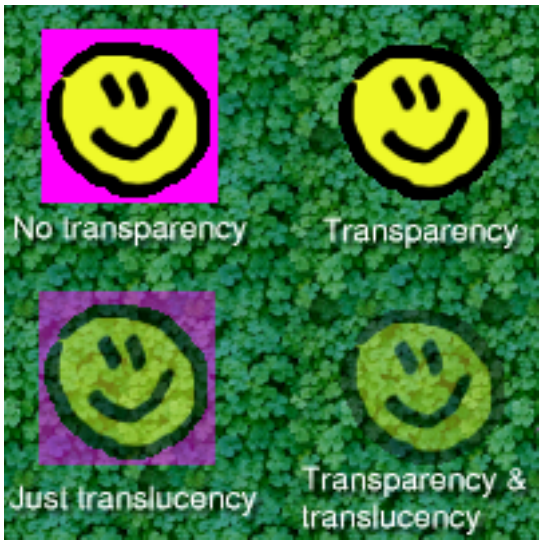

Figure 1: transparency and translucency

TIP **Avoiding Pink Noise**

As transparent pixles must have exactly the colour FF00FF you often get pink noise around the edges of your image since many gfx apps will use anti-aliasing which creates several not-quite magenta pixels. Here are a few ways to avoid this:

- In Photoshop or GIMP make sure to set the tolerance of tools such as the magic wand or the fill bucket to 0 and to turn anti-aliasing 'off'. If you do not do this you will often inadvertantly create pink noise when using such tools on the outer edges of your graphic.
- When creating the gfx do them on a transparent background (or layer). When you are done convert the colour depth to 'indexed'. This only allows for transparent or non-transparent pixels and therefore removes any anti-aliasing at the edges. Then convert back to RGB and flatten the image using a magenta background (See figures below).

2.3 Animations

Sometimes you will want gfx to be animated. For example a mapobject (or a part of it) like a flame may be animated. Adonthell's own gfx format supports animations (also like GIFs!). You can create them by making each frame as a seperate PNM (if some frames get repeated you only need to make them once). The frames can then be loaded into the developent utility (animedit??) which comes with Adonthell. You can then put the frames into the desired order and set the duration for which each frame should be supplied. You can create animations that loop for ever and ones that just play once and then stop.

Once saved the complete animation can be loaded by Adonthell's other editors to be included into mapobjects or cutscenes. Note that mapcharacters are animated in a different way because they need to be interative. To find out how they work please read chapter 4.2.

Adonthell specific

2.4 How to use the Adonthell data CVS

We use a CVS to store all of Adonthell's gfx centrally. Basically there is an online copy of everything we have. With the appropriate utilities you can then download a copy to your computer with which you can then work offline (this is known as 'checking out the CVS'). Any new gfx you make you can mark to be added to the central CVS. You may then do a 'CVS update' which uploads your new work and downloads any changes other people have made. That way all developers can stay in-sync with each other. CVS also allows us to undo changes incase someone has made a mistake and various other maintenance features.

To use this you will need a CVS client program (they are available for most Operating Systems) and you must also be registered as a developer with our Savannah project (http://savannah.gnu.org). You can then connect to our CVS with the following details:

bla bla bla