Graphic Compression in Retro Video Games

A report submitted to the Cluster Innovation Center in partial fulfillment of the requirements for paper

VI.2 Computer Graphics and Visualization Architecture

(Semester VI)

Ankit (Roll No. 11706)

Prabhakar Deep Tirkey (Roll No. 11730)

Yatharth Rai (Roll No. 11745)

Under the supervision of

Mr. Neeraj Kohli

Cluster Innovation Centre

University of Delhi

Delhi-110007

Certificate

This is to certify that the present work embodied in this report entitled “Graphic Compression in Retro Video Games” has been submitted to Cluster Innovation Centre, University of Delhi towards the partial fulfillment of paper VI.2 Computer Graphics and Visualization Architecture. This project has not been submitted in part or full to any other University/Institution for the award of any other degree or diploma. The work has been carried out at Cluster Innovation Centre under the supervision of Mr. Neeraj Kohli. However, the supervisor does not undertake any responsibility for the authenticity of the text.

Acknowledgment

With a deep sense of gratitude, we express our dearest indebtedness to **Mr. Neeraj Kohli** for his overwhelming interest in our project. We would like to thank him for giving us the opportunity to do this wonderful project - **Graphic Compression in Retro Video Games** His learned advice and constant encouragement have helped us complete this project. This is our proud privilege to be his student.

Abstract

Video Games have been around for a few decades now. Hardware limitations, and the lack of growth in the industry in general caused video game companies to run on a certain budget. In the 1980s, a new form of animation, called sprites was developed. These involved making multiple frames of a single object to cover the different animations. These sprites by themselves were not large, but could not be used until they were compressed due to memory complications. Therefore, several workarounds were deployed. Sprite compression involves decomposing a larger, meta-sprite into smaller sprites, and containing them within the graphics table. Decomposition of the sprites is a manual process. We aim to automate the decomposition to make it faster. This technique could also be used for several different types of data compression, by exploiting the repetitiveness of the data in general.

Table of Contents

List of Figures

Introduction

Computer graphics is a branch of computer science that deals with generating images with the aid of computers. Today, computer graphics is a core technology in digital photography, film, video games, cell phone and computer displays, and many specialized applications. A great deal of specialized hardware and software has been developed, with the displays of most devices being driven by computer graphics hardware. It is a vast and recently developed area of computer science. The phrase was coined in 1960 by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, or typically in the context of film as CGI. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the consumer. It is also used for processing image data received from the physical world. Computer graphics development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design in general. Computers have been around since 1936. Although, not at all equivalent to the personal computers which we use as of now, computers back then were more or less similar to calculators, capable of performing complex arithmetic operations. Over time, as computers began to get more accessible, and available everywhere. The first video-game developed was *Pong* in 1972. As miniaturization of components became possible, computers began to be used as personal machines, and over time, video game development took off as more powerful hardware was made accessible to the people. Graphics in video games has evolved rapidly in the past decades, as more powerful hardware was made available, and computation power increased. Video Games now tend to occupy more than 30GBs of memory through their assets, but in the late 1990s, videogames were developed to be as small as possible. The 1990s was the third decade in the industry's history. It was a decade of marked innovation in video gaming. It was a decade of transition from sprite-based graphics to full-fledged 3D graphics and it gave rise to several genres of video games including, but not limited to, the first person shooter, real-time strategy, survival horror, and MMO.

Sprite Based Video Games

Animation, in a simpler essence refers to a collection of moving imagery. In early in the 90's decade, and before, most of the video games developed followed a certain form of animation called sprite animation. This involved creating multiple frames of an object, as it performed a certain action. In computer graphics, a sprite is a two-dimensional bitmap that is integrated into a larger scene, most often in a 2D video game. Sprites were developed at Texas Instruments by Daniel Hillis. Originally sprites referred to independent objects that are composed together, by hardware, with other elements such as a background. The composition occurs as each scan line is prepared for the video output device, such as a CRT, without involvement of the main CPU and without the need for a full-screen frame buffer. Sprites can be positioned or altered by setting attributes used during the hardware composition process. Examples of systems with hardware sprites include the Texas Instruments TI-99/4A, Atari 8-bit family, Commodore 64, Amiga, Nintendo Entertainment System, Sega Genesis, and many coin-operated arcade machines of the 1980s. Sprite hardware varies in how many sprites are supported, how many can be displayed on a single scan line (which is often a lower number), the dimensions and colors of each sprite, and special effects such as scaling or reporting pixel-precise overlap. Use of the term sprite has expanded to refer to any two-dimensional bitmap used as part of a graphics display, even if drawn into a frame buffer (by either software or a GPU) instead of being composed on-the-fly at display time. The act of manually creating sprites, as opposed to pre-rendering them or using digitized images, is a form of pixel art. It is sometimes referred to as spriting, especially in the hobbyist community. The use of sprites originated with arcade games. The first video game to represent player characters as human player images was Taito's Basketball, which was licensed in February 1974 to Midway, releasing it as TV Basketball in North America.

Figure 1: RPG Maker MV Character Sprite

Sprites are two-dimensional images or animations overlaid into a scene. They are the non-static elements within a 2D game, moving independently of the background. Often used to represent player-controlled characters, props, enemy units, etc., sprites can be composed of multiple tiles or smaller sprites. They can also be used for pseudo-3D sprite scaling like in Super Scaler, Mode 7, or in pre-rendered movement. More recently, sprites are also being used in web design (like the CSS sprites which combine lots of small images into one sheet to improve web page load times). The image shows how a sprite based animation worked. There are multiple images covering the character as it walks left, or right, or falls up or down. The constraint with such a form of animation was that it couldn't cover more dynamic forms of movement, as every single frame needed to be made, it took time and manpower. Therefore, these games had a limited sense of movement, but it does not deter the fact that a lot of good games were masterpieces of their time.

Hardware Limitations

One of the most popular video game consoles of the 1980s was *Nintendo Entertainment System*, popularly called as NES. It was an exported version of FC or *Famicon* as it was known in Japan. It was responsible for revitalizing the videogame crash of 1983.

Figure 2: Famicon by Nintendo Entertainment System

NES was a pinnacle at gaming consoles then, consisting of Ricoh 2A03, an 8-bit microprocessor containing a second source MOS Technology 6502 core, running at 1.79 MHz for the NTSC NES and 1.66 MHz for the PAL version, and 2kB of on-board RAM. The graphics are categorized into 2 - background, and foreground. Background graphics consisted of data that did not move, hence stayed ‘static’. Foreground graphics referred to the moving parts of the game, which are called sprites.

Each set of categories could hold up to 256 graphical 8x8 pixel tiles. A single tile can hold upto 3 different colors at once. The NES was limited to using 4 different sprite palettes at any given

time. In NES, a single scan line could only hold upto 8 sprites at any given time. Any excess sprites would not be rendered. A scan line (also scanline) is one line, or row, in a raster scanning pattern, such as a line of video on a cathode ray tube (CRT) display of a television set or computer monitor. On CRT screens the horizontal scan lines are visually discernible, even when viewed from a distance, as alternating colored lines and black lines, especially when a progressive scan signal with below maximum vertical resolution is displayed. This is sometimes used today as a visual effect in computer graphics. The term is used, by analogy, for a single row of pixels in a raster graphics image. Scan lines are important in representations of image data, because many image file formats have special rules for data at the end of a scan line. For example, there may be a rule that each scan line starts on a particular boundary (such as a byte or word; see for example BMP file format). This means that even otherwise compatible raster data may need to be analyzed at the level of scan lines in order to convert between formats.

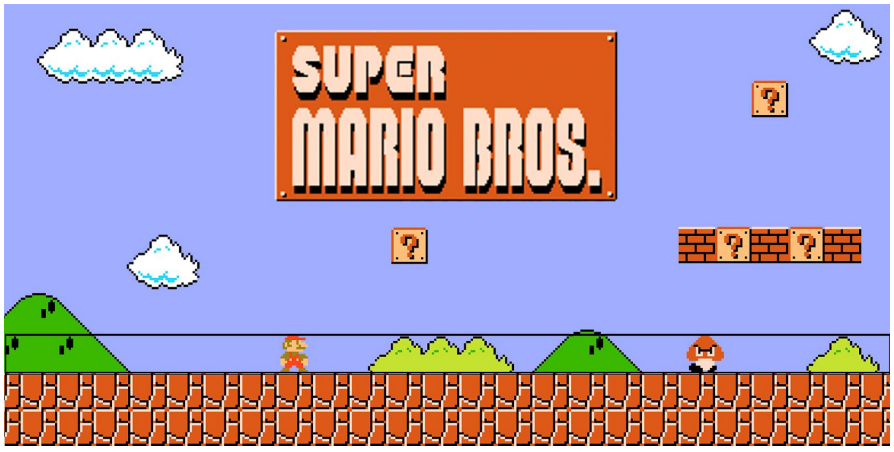

Figure 3: Super Mario Bros. Game

The consoles, by themselves carried no video game ROMs, but required the user to buy a cartridge. The games were written on *6502 ASM*. These cartridges had 8kB reserved for graphics, and 32kB reserved for code and data.

Each SNES sprite can have 16 colors and a palette slot out of 8 total palette slots. Disregarding advanced tricks like rewriting palette data during a scanline, this is the general limit. The 0th entry of each palette slot is transparent, regardless of what color is specified there. A SNES sprite

can be 8x8, 16x16, 32x32 or 64x64. Sizes cannot be chosen freely. A game can have two of the predetermined size combinations: 8x8, 16x16 8x8, 32,32 8x8, 64x64 16x16, 32x32 16x16, 64x64 32x32, 64x64 This means that, once picked, the NES will consider every sprite in the game to be one of these sizes.Thus, if a game is using 8x8 and 16x16 sprites, a 32x36 character would actually be considered as 3 sprites. If the game was using 32x32 and 64x64 sprites, then the 32x36 character would be 1 sprite. Sprites per Scanline: 32. This is a hard limit. On top of that only 34 8x8 tiles can be displayed regardless of how big the sprites are. This is related to the next limit. No matter how many sprites are there, after 256 pixels the PPU starts cutting off sprites. The renderer always clips the frontmost (lower id) sprites. Sprite VRAM Size: There are two 128x128 pixel areas allocated for VRAM. Each of them are assigned to... nothing actually. Each sprite can be told which size it should take and where from VRAM it should be taken from. There is a flag for each sprite for which VRAM area it should be fetched from. 128. That's how many simultaneous entries there are for sprites.

Graphical Workarounds

Most of the graphics ingame are taken by sprites, which refer to the ‘moving’ part of the graphics. The size of sprites depends, but a larger sprite is called metasprite. In essence, a single, larger sprite can be broken into multiple, smaller sprites. This is one of the most common techniques used to compress a certain sprite. It can be shown as the following.

Figure 4: Distribute all the sprites in the table one to one

These are sprites for a character in an NES video game. These blocks are larger, and occupy 16 blocks of memory. Since a normal NES game only has 256 blocks to store sprites, containing 15

different animation sprites would result in 190 blocks used, which is a lot for a single character. Therefore, these sprites are decomposed into smaller, subsequent blocks.

Figure 5: Sprites are decomposed into smaller parts

The sprite has been divided into three parts. The face remains constant. The only changing parts of the following sprites is the head and the cowl. Now, looking at the head, there are only a few minute changes in the head sprite, therefore, we can decompose the following sprite further.

Figure 6: Sprites images are divided in smaller parts

The left portion of each sprite remains the same, the only thing that changes is the left. Therefore, it is safe to take the left portion of the sprite and re-use it. Similarly, the left and right of the first and the last sprite, are mirror of each other, therefore, we can lessen the size again by mirroring the same sprite. If we divide the pictures with the hairstyle into separate squares, then we will notice that they have repeating parts. Therefore, it is enough to draw one detail, and then use it in all three hairstyles.

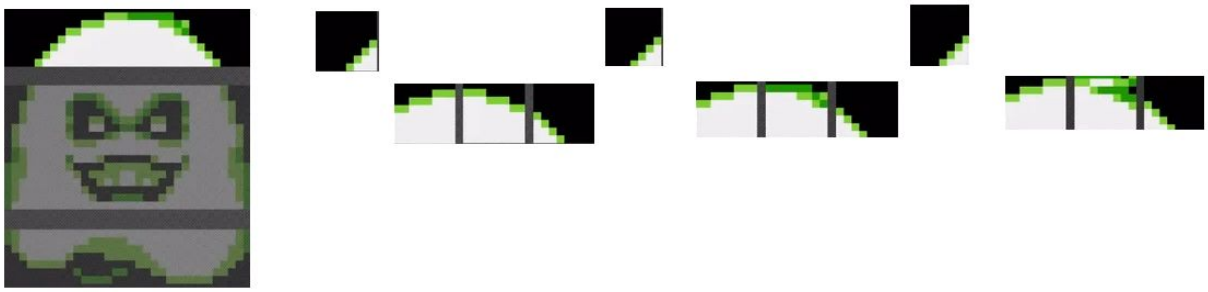

Figure 7: They have repeating parts

Another similar technique is Sprite Flickering. Sprite flickering on the NES would occur when more than 8 sprites were displayed on the same horizontal line. We kept the sprite count as low as we could, but as previously mentioned, we didn't sweat the exact numbers. Some of our objects produce a few more particles than an NES game would allow. Some games like Recca or Contra got around sprite limits by displaying certain sprites only every other frame (at 30fps instead of 60fps). On CRT monitors running low resolution interlaced video, objects would appear to be drawn every frame. In addition to this, NES particle art was often built with flickering in mind for effects like explosions.

We used flashing sprites in some situations to replace alpha transparency; For example, Shovel Knight flashes on and off during his "invincible" state, after being hit. So overall, this didn't feel like an important restriction to follow, unless it made the gameplay not feel like NES gameplay. Color Palette Additions The NES was only capable of spitting out 54 different colors. There isn’t a very useful yellow, the darker spectrum of color is very underrepresented, and there aren’t many shades that work for displaying a character with a darker skin tone.

Automation

These are a few tricks that are used to reduce an image in an NES game. Most of the following is done manually, although it is not hard to code for the following. We have accomplished the following automation through the workflow as described below -

1. Split the images into an n x n matrix. (Note - n is a power of 2) It is preferable to split into an 8 x 8.
2. Compare the images one by one.
3. Flip the images and compare again.
4. Create a larger binary coded database to mark the sprites.

Upon comparing our results to the memory it would've taken were the sprites not decomposed, we reduced the entirety of sprites by 86%. The compression means that instead of taking 190 blocks, we compressed it into a mere 26 blocks. Such a compression is extremely useful to represent larger sprites.

Demonstration -

Figure 8: Cacodemon from DOOM

The above image is a Cacodemon from the DOOM series. Although, the automation requires the user to enter the number, intuitively, we can split the following into 4 parts. Using the script on the following result was generated.

Figure 9: Splitted Cacodemon in 4 parts

Using the sprite compression module, the entire sprite is reduced to 2 separate sprites, due to the fact that one side is a mirror of the other side.

Figure 10: Using Compression module, Cacodemon is reduced to 2 sprites

Therefore, resulting in the compression of the following by 50%. Larger sprite sets, such as the Boo sprite set mentioned in the previous section can benefit more from the following compression algorithm. Even though not perfect, the following helps in reducing the amount of manual workload a sprite animator will go through in a certain optimization time.

References

1. “Image Compression Strategies For Game Developers.” blog.100tb.com, May 3, 2019. https://blog.100tb.com/image-compression-strategies-for-game-developers. [Accessed May 20th, 2020]
2. “Keeping the Game Alive: Evaluating Strategies for the Preservation of Console Video Games” by Mark Guttenbrunner, Christoph Becker, Andreas Rauber, Vienna University of Technology https://publik.tuwien.ac.at/files/PubDat_190893.pdf [Accessed May 20th, 2020]
3. “Как Писали Игры Для Приставок: Чудеса Оптимизации и Жесткий Кодинг.” Яндекс Дзен | Платформа для авторов, издателей и брендов, https://zen.yandex.ru/media/code/kak-pisali-igry-dlia-pristavok-chudesa-optimizacii-i-jestkii-koding-5e16ead7118d7f00adfc20a7. [Accessed May 20th, 2020]