Godot: In The Eye Of The Beholder

A Review By Todd S. Elliott

In this second issue of ***Commodore Currents***, I wish to shine the spotlight on **GoDot**, which is still actively developed and supported by its author, **Arndt Dettke**. Graciously, **Arndt Dettke** has assisted in proofreading the review for technical accuracy. Thanks goes to **Bruce Thomas** for additional proofreading.

Just what is ***Commodore Currents*** all about? This publication is primarily a vehicle for reprinting all of my articles I've written for various **Commodore** oriented publications. In a true Commodore hobbyist fashion, I publish this endeavor via **Wheels**, **geoPublish**, **geoWrite** and **PostPrint** II/III and use the power of **PostScript**. However, I will be using a mainstream computer to do some graphics, such as the ***Commodore Currents*** logo, grabbing screenshots of actual Commodore 64/128 screens, **GhostScript** for the PDF conversion, **GhostView** for the layout proofing. *This is **Commodore GEOS** printing at its finest!*

For this issue, I've selected a comprehensive review I've originally submitted to the July 2001 ***LUCKY Report***, one of the premier Commodore GEOS publishing efforts edited by **K. Dale Sidebottom**. The ***LUCKY Report*** has ceased publication, as Dale and **Rolf L. Miller** of the **UCUGA** users group now undertake the ***Commodore Digest*** (Commodore GEOS) publishing endeavor.

In 2003, there is quite a bit of Commodore hobbyist activity going on regarding **GoDot**. In addition to programming new **GoDot** modules and bug-fixing **GoDot**, **Arndt Dettke** (and other authors) have begun to produce **GoDot** tutorials for publication within the ***Commodore Digest***. For a nominal fee, The **LUCKI Club** will distribute a complete **GoDot** disk for those who prefer the convenience of having the real disk media instead of downloading it.

For some time now, **Matthias Matting** has supported **GoDot's** 4bit files for his **ConGo** utility for mainstream computers running in the **Windows** environment. There is a program called **XnView** for mainstream computers running Windows and **LINUX**, that can also view 4bit files and convert them to modern graphical formats.

Continuing the 4bit theme, I have developed a **Wheels** (GEOS) utility, called **dotView**, which can view 4bit files in multicolor graphics splendor. Please check out the bibliography at the end for website links to these programs and more.

About the future of this publication from the editor: ***Commodore Currents***, for the next few issues, will contain reprints from my Commodore article repository. While I am not as prolific as other notable Commodore scribes, I hope to organize my content over several issues on a semi-regular basis. For the time being, ***Commodore Currents*** will be made available as a PDF file, freely downloadable on the **Internet**.

Once I have a 11x17 printer, I will seriously examine publishing possibilities for a newsletter or magazine giving the usual breadth and depth of leading Commodore concepts out there. Such coverage, not necessarily limited to, would emcompass GEOS, demos, programming, with an eye towards using the machines as a main hobby instead of waxing a nostalgic look like those 'retro' magazines usually do. Of course, since this would be published in-house, there's no printer delays, etc. Still, this publication would be a hobbyist endeavor and the promise of a prompt publishing schedule is one that cannot be kept, hence I hope to undertake it on a semi-regular basis.

This publication, while not perfect, strives to publish material that recognizes all relevant intellectual properties as belonging to their respective authors. This publication recognizes all valid trademarks as belonging to their respective holders and no infringement is intended.

To the extent practical, I have secured some permissions from the author(s) to reprint the relevant works here. If you, the reader, possess an intellectual property interest in any material presented in this publication, please let me know of any objection(s) you may have.

I have included a short bibliography here to give due credit(s) to various authors and other program(s) that have contributed to the successful creation of this publication.

Wheels OS
SuperCPU
All by **Maurice Randall** at:
www.cmdrkey.com
geoWrite v2.2
geoPublish v1.1
All original code by **Berkeley Softworks** and licensed to **CMDRKEY**. All code modifications done by **Todd S. Elliott**.
GoDot
By **Arndt Dettke** and **Wolfgang Kling**. Arndt now maintains **GoDot**.
www.godot64.de
GhostScript/GhostView
www.cs.wisc.edu/~ghost/

(*Continued on Page 10*)

The GoDot Report

Copyright (C) 2001 by Todd S. Elliott

The Past and Present of Commodore Computer Graphics

One of the **Commodore 64**'s selling points back in its halcyon days of the Eighties was the fact that it had excellent color graphics. Almost immediately after the Commodore 64's introduction, graphic painting programs began to appear, most notably **Doodle!**, **KoalaPainter** and **geoPaint**. With all of these painting tools at a user's disposal, Commodore 64 users began to create many screen masterpieces.

Commodore magazines at the time held contests for best artistic creations for the Commodore 64 and even held regular monthly galleries of graphical pictures. Many people made a name for themselves in Commodoredom for their pixel artistry such as **Wayne Schmidt**, **Walt Harned** and **Daniel 'DeeKay' Kottmair**.

That was then; This is now. The art of computer graphics have evolved and expanded in unforeseen directions in the past twenty years of the computer evolution. Gone are the days of *drawing* and creating computer graphics. Rather, computer users of modern computing platforms are *processing* graphics instead of creating them.

Before I lament the lost art of creating computer graphics, it is nigh time that computers are finally harnessed with their vast computing potential in processing such graphical creations. Indeed, it is difficult to create graphics freehand just using a mouse, joystick and/or keyboard. It is far more simpler to process existing graphics and in doing so, fully maximizes and taxes such computer hardware.

Waiting for GoDot

Ten years ago, such a program was created for the Commodore 64 that allowed the users to process graphics as well as create them. It is called **GoDot** and was released on a piecemeal basis in the pages of the German magazine, **64'er**. Each issue of 64'er would have a module or two for GoDot and would have a tutorial as well. Over a period of time, GoDot was a fully realized program for the Commodore 64 that was rich in features and abilities.

Modern day computing platforms are more often 'processing' graphics rather than 'creating' them.

But, many users in North America have never heard of it or did not fully appreciate the power that carries with image processing. We were still creating graphics, most often laboriously by hand via the mouse/joystick as opposed to processing them. Thankfully, **CMD** realized the marketing potential of a fully featured program and made plans to market it to the North American Commodore public.

Amazingly, despite its launch in the pages of the German 64'er magazine, GoDot's user interface was written in English, no work was needed to be done for the North American audience. The author, **Arndt Dettke**, had big dreams for GoDot, intending it to be used all over the world.

However, an English manual was needed in order to market such a program. I do not mean to disparage CMD; Obviously, they made a good effort in porting English documentation for GoDot and producing it in their own style. But they only made a perfunctory manual, describing the functions of the GoDot main user interface and a library listing of modules germane to GoDot. Oh, there was a three page tutorial.

Where were all of those supplemental prose extolling the virtues of image processing in lieu of image creation? Where were illustrations, screen shots and writings which would inspire creativity in a Commodore user to immediately fire up GoDot and process/create graphics? Where were all of those tutorial articles that were piecemealed through the pages of the 64'er magazine? Arndt Dettke stated that he would have to translate many of the tutorials into English and such work would have raised the cost substantially for CMD in marketing such a program. Such functional and business-like documentation has led to several misconceptions about GoDot which has effectively impeded its acceptance in the North American classic Commodore community.

Deconstructing the Myth of GoDot

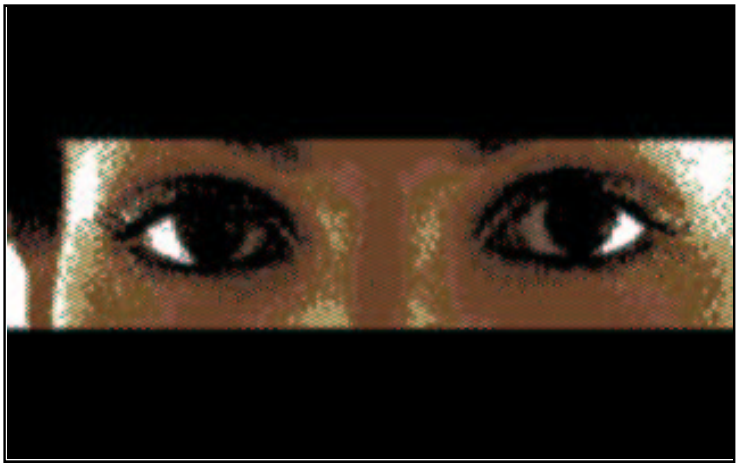
First of all, GoDot, while it can create graphics, is not really suited for this kind of task. In fact, a module called **PixelEdit** was released nearly ten years after GoDot's inception and was produced only at the urging of a North American Commodore user, **Hugh McMenamin**. There are many drawing programs that will run (draw!) circles around GoDot in creating graphics.

Secondly, another misconception about GoDot is that it can only convert graphics. This was my conclusion and observations about GoDot and its CMD marketed English manual. I could load in graphical formats that were commonly found on other modern computing platforms and convert them to the interlaced resolution format of our Commodores, despite the fact GoDot can do much more than that. Again, there are dedicated programs for the Commodore 64 that can do the conversion job like **geoGIF**, **iPort** for the 128, for example.

Rather, GoDot is an *image processor* above and beyond anything else. It can process existing graphics, modify them, apply complex mathematical formulas, create effects and many more. While not in the class of **Photoshop** (for **Windows/Mac**) or **GIMP** (for **LINUX**), GoDot does the job very well! GoDot, not surprisingly, given its enormous task in processing graphics, benefits greatly from the increased computing power of the **SuperCPU**.

With respect to the North American Commodore userbase, we have been creating graphics for so long that all of a sudden, a very 'new' concept of processing graphics is now present in such a program like GoDot. There has been some growing pains and some Commodore users began to accept such a paradigm shift towards processing graphics on their classic computers. The acceptance of such a program has been very slow and has baffled GoDot's main advocate, Arndt Dettke. Despite that, Arndt Dettke has gone beyond the call of duty by offering excellent support and rapport with many of the North American users by creating many modules for GoDot based on our suggestions.

Like Postscript printing on the Commodore, it truly has taken a full decade for the North American classic Commodore computing scene to begin warming up for such revolutionary changes and concepts in how they operate their computers. GoDot is of no exception and I feel that this program is truly one of a kind for the Commodore.



Left: GoDot as it appears in the eye of the beholder.

Bottom right: GoDot, a sleeping beauty?

GoDot, while it can create graphics, is not really suited for the task. In fact, the only module designed to create an image is called mod.PixelEdit.

The nitty-gritty details behind the images

Let's get into the inner workings of GoDot so that we all can have a better understanding as to what image processing is all about and why it is important for the Commodore 64/128 computers to enjoy such benefits. The Commodore 64 computer itself has its own graphical bitmap formats; Yet, strangely enough, GoDot, a native Commodore 64 program, eschews such formats and implements its own format for internal image processing operations, the **4bit** graphical format. You could be more accurate to say that GoDot is a 4bit image processor which runs on the Commodore 64 rather than a Commodore graphic image processor!

This 4bit format is the linchpin of which GoDot bases its entire foundation for image processing. The 4bit format merely shows the brightness levels of the graphical bitmap image. All pixels in a 4bit bitmap image are always *on* and have a brightness level assigned instead. In a conventional Commodore graphical bitmap, a pixel is where it's *on* and has a color assigned to it and appears onscreen, and a pixel where it's *off* would not appear onscreen. This is not the case with GoDot.

This brightness level within the 4bit format is limited to sixteen different levels, going from a value of 0 (black) all the way to 15 (white). It is on a luminance basis with colors ordered on according to their luminance values. This is also similar to a gray scale, where if you use a B&W monitor hooked up to a Commodore 64 using GoDot and viewed its palette, you would see these colors indexed with varying shades of gray. This brightness scale is smooth and pleasing in appearance to the naked eye. Keep in mind that this is an internal format for GoDot's own use.

One final word about this brightness scale; Have you ever wondered how to describe a color to someone who has never seen them before or are totally blind? You would resort to analogies or known references in trying to describe such colors. The computer is of no exception. It does not know any colors. What the computer does understand is mathematics. Mathematics is commonly used as a very

effective tool in describing the unknown in our modern world. This brightness scale is fully indexed from 0 to 15 and can be easily manipulated and understood by the computer with sheer mathematical brute force.

GoDot in action

Now, this is where I hope to unlock your creativity and force you to think out of the *bread*box, go beyond your Commodore's abilities in doing so. The Commodore computer is a computer with many possibilities for input and output. One main output is the screen, of course and let's focus our energies there.

The standard Commodore multicolor screen that is commonplace in many games, demos and painting programs is a very simple graphical format with many restrictions and tradeoffs in achieving the desired graphical effect. This screen can only support up to 4 colors in a single 4 pixel by 8 pixel cell on a screen. The problem is that the internal 4bit format can support up to sixteen brightness values (represented by colors) in a single 4x8 cell on a screen! How is GoDot going to display the contents of the 4bit data onscreen in a multicolor format?

First phase ---> Optimization

First of all, GoDot tries to optimize brightness levels in a single 4x8 cell. It does so by creating a *histogram* or a running count of brightness levels found in a 4bit file. By process of elimination, it chooses a single color that best supports the entire multicolor screen. This color would be assigned to the background color. This is done on the first pass through the 4bit file. On the second pass, further color optimization & counting is done on each and every 4x8 pixel cell to come up with three more color combinations unique to that cell. There are 1000 such cells to process. All of this 'brightness' counting and building the histogram takes place internally.

Computers do not know any colors and only understand mathematics. Mathematics is used as a very effective tool in describing the unknown in our modern world.

Second phase ---> Error Correction

GoDot then attempts to display the 4bit data directly to the multicolor screen on your Commodore. On this second pass through the 4bit file, GoDot analyzes each 4bit data it goes through and assigns to them pixel values of which would be outputted towards the screen. It compares the 4bit data that is currently being analyzed against the 4 optimized brightness values it had already calculated in a histogram table unique to that 4x8 pixel cell. If there is a direct match, a pixel value is generated. But, what if there is no match? Remember, a 4bit data file can have 16 brightness values in a 4x8 cell as opposed to the Commodore's multicolor standard of 4 colors in a 4x8 cell. So, there will be a loss in color depth in such analysis, where there are images in the 4bit data that provides more than 4 colors per cell.

This analysis stage is what I like to call the *error correction* stage. As GoDot progresses through the 4bit datafile, it will encounter such color errors due to 4bit values being analyzed and that they are not one of the four most optimized values. Such errors cannot be ignored and GoDot tries to correct such errors throughout its analysis of the 4bit data. But, just how does such error correction occur? Would the human eye notice?

This is one of the principal reasons why GoDot adheres to a brightness scale of 0 to 15 instead of conforming itself to stock Commodore colors. In correcting such errors, GoDot uses mathematics in arriving at the *nearest* valid brightness value from the 4 brightness values as found in the optimized histogram to replace the offending 4bit data being analyzed. This brightness scale ensures that such error correction would be eye pleasing and smooth as much as possible. If the stock Commodore color scale of 0 to 15 were used, such error correction would not conform to a eye-pleasing brightness scale, but to a color scale that was designed by Commodore back in 1982 and would yield poor results and those infamous color clashes.

Third phase ---> Renderizing

The multicolor screen has been changed considerably and is now full with pixel values making up its entire onscreen bitmap. GoDot now enters the renderizing phase and actually substitutes the brightness values with their respective Commodore colors, using only the four brightness value optimizations from the histogram table for each and every individual 4x8 pixel cell. There are 1000 such cells, and it takes up some time in GoDot going through all of them and finally displaying the screen in its full multicolor glory. With most 4bit files, the final results that GoDot renderizes onscreen is surprisingly very good. Throughout the entire displaying process, GoDot has never changed the 4bit data area one bit.

What a magic trick it is!

Can you repeat, please?

GoDot only has made such mathematical gyrations with that particular 4bit file to the multicolor screen with surprisingly good results. But, GoDot can increase its power by changing its computational algorithms on a different level. Let's say that you want to create a file that can be displayed in the interlaced resolution format of our Commodores. This interlaced format is the one you can see running in demos, **Steve Judd/Adrian Gonzales**' JPEG viewer and my standalone **NTSC IFLIVIEWER**. This interlaced format has less restrictions than the stock multicolor format and can display up to 136 colors in a 296x200 screen resolution. GoDot cannot display this enhanced video mode, so it needs to save the renderizing results to a standalone file.

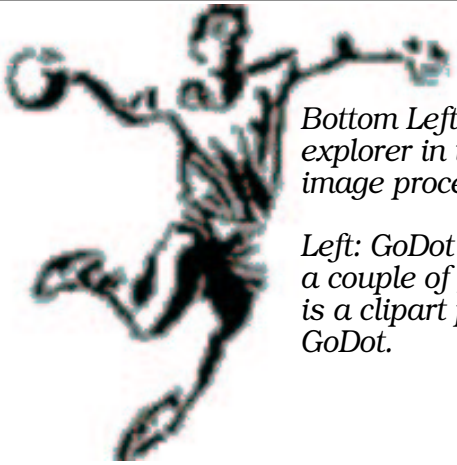
GoDot applies the same mathematical discipline to the same 4bit datafile as it had done previously in generating a multicolor screen, but now for this interlaced resolution format. This time, the optimization phase of the 4bit file analysis is expanded to include the increased color selection and increased resolution. Error correction still occurs and pixel/color data is created to a standalone file for later viewing by an external application.

Let's say that you want to create a **.GIF** graphics file instead and use the same 4bit datafile. GoDot applies the same mathematical controls as used previously. But a .GIF file can exceed the color depth that is possible in a 4bit file. GoDot merely makes an identical copy of the 4bit data area into a .GIF bitmap without any processing and error correction. Then you can send this .GIF file to someone on a modern computing platform. You can even set up your webpages with .GIF graphics processed by GoDot. The GIF saver module has recently been greatly improved, with a new user interface, some bugs fixed and the ability to save a clipped area as a standalone GIF.

If the stock Commodore color scale of 0 to 15 were used, such error correction would not conform to an eye-pleasing brightness scale.

Let's say that you want to create an .EPS file, which is shorthand for **Encapsulated PostScript**. It is similar to a geoPaint photoscrap, but for the PostScript world. GoDot will run through the same 4bit datafile, and create an exact copy of the 4bit data into this EPS bitmap without any processing and error correction. Then you can use this .EPS file on modern computing platforms or right here on our Commodore, by using the **PostPrint** program created by **Maurice Randall**. PostScript printing has never looked so good!

GoDot enjoys a high degree of device independence akin to PostScript. All it does is to deal with graphical images and manipulates them with sheer mathematical precision. It can accept input from a wide variety of sources like .GIF's, native Commodore graphical formats, **HandyScanner**, etc. It can then format output for .GIF's, native Commodore graphical formats, printers, the World Wide Web, etc. GoDot's conceptual framework takes care of all the differences in between those disparate inputs and outputs of which the Commodore can undertake.



Bottom Left: GoDot, a space age explorer in the modern era of image processing.

Left: GoDot plays handball with a couple of graphics cards. This is a clipart picture converted by GoDot.

Show me what GoDot *really* can do

While it is great to receive graphical information from just about any input and prepare such graphical information for just about any output on your Commodore, it's what you do in between which makes GoDot earn its stripes as an image processor. GoDot comes with a slew of processing modules that affect the 4bit data area ranging from dedicated error correction modules to color/dithering modules to modules that act like camera filters. There is also a color controls section which affects GoDot's renderizer routines, but not the 4bit data itself.

A Mini Walkthrough

Let's say that you come across a file on the Internet that you want to play around with. For example, the *Wave Friendly* logo looks like a nice candidate. It's in a .GIF file, so GoDot doesn't have a problem loading it in. I use the **Color Controls** part of GoDot to change the overall color of the wave in the 'Wave Friendly' logo to a color more to my liking. This is just one small part of the Color Controls as it can also control the brightness of the image, use dithering/patterns to come up with eye-pleasing color combinations and more.

Next, I turn my focus on the **ClipWorks** module. The 'Wave Friendly' logo is small enough so that it can be reproduced across the screen. I execute the clipping module and clip out the picture containing the logo, similar to what a person would do in geoPaint. Using a **TileClip** module, I make a copy and *paste* it all over the onscreen canvas, creating a tiled effect of the 'Wave Friendly' logo. **Andy Warhol** would be so proud.

GoDot has made such mathematical gyrations with a 4bit file with surprisingly good results. GoDot can increase its power by changing its computational algorithms on a different level.

There is a lot more to the ClipWorks module and its framework of related modules like TileClip, as it can allow a person to squeeze (**Squeeze2Clip**) or stretch (**StretchClip**) a graphic that is contained within the clipping area. The clipped area can be flipped (**UpsideDown**) and mirrored (**Mirror**). GoDot can also restrict further image processing options to the clipped region only. GoDot can allow for *masking*, overlaying a clipped image onto another image.

Next, I apply one of those convolution modules contained within GoDot. They're similar to filters on a camera which allows for some unique effects. There are a lot of options here, from blurring an area, applying specific textures, creating cartoonish effects, etc.

I apply two effects, the first one was called **DeepPress** and it created a 3-D impression. The second effect was called LaPlacian and it created a neon effect. I found this neon effect by accident as I was simply trying out modules and seeing what results I could come up with.

I'm pretty much done with the image processing of this 'Wave Friendly' logo and there are many options at this point. I could save the file as it is, a 4bit file. I'll do that so I can always refer to it again in creating different kinds of output. Next, I could save it as a .GIF file and post it on a website. I could create an **IFLI** file and view it in its full interlaced glory. I could print it out. GoDot supports nearly every printer imaginable, from the **HP** brands, to **Epson** brands and the **Canon** brands. Recently, in a marriage of two technologies, GoDot supports the EPS (Encapsulated PostScript) file format for laser printer output. I elect to save the file as an EPS file, and this file can be pasted into a newsletter such as the **LUCKY Report** as if it were a photo scrap. Printer output is the best and closest way in what GoDot provides for image data material as no processing, renderizing, or error correction takes place.

Under the statistical microcosm

The main thrust of this seemingly simple exercise in using GoDot shows that I haven't *created* anything from scratch using just my mouse. I used the Commodore 64 and its computing power, even at 1MHz or 20MHz, and processed such graphics in a purely mathematical fashion. For example, in the color controls section, GoDot counted the colors and made the appropriate changes in color selection when I changed the palette. I used the ClipWorks module and TileClip

module in creating a tiled effect of the 'Wave Friendly' logo and the computer used brute copying mechanisms to copy a clipped area all over the screen. Next, when I applied the DeepPress and LaPlacian effects, GoDot used statistical sampling algorithms in achieving the desired look. Next, when I saved it to an EPS file, GoDot made an exact copy of the 4bit data area into a bitmap and covered it with the relevant EPS code.

A building in Amalienburg, Germany.

All graphics accompanying this article were converted by ConGo from 4bit files.

Using GoDot does not make Graphical Gurus out of you...

Admittedly, GoDot is a very complex piece of software rich in features and options. There are a lot of modules that allow near unlimited creativity on the part of the Commodore user in processing graphics. However, a person who uses GoDot does not automatically become an artist or create graphical masterpieces overnight. GoDot provides a lot of tools at the user's disposal and the user needs experience in trying them out, pushing them to the limits and push the creative envelope further. Moreover, with the huge array of tools, it only makes sense for the user to plan things out with respect to image processing & creation.

Just use GoDot and plan ahead of what you are trying to do with an image. Are you designing graphics for the World Wide Web? Are you preparing graphics for printed output? Are you just creating graphics for the Commodore's varied resolution modes? Then try out various tools, observe what happens, accumulate invaluable experience. Be sure to have the *Undo* function handy, so you can reverse any mistakes you may make. If you have the Go64! issues, check out the GoDot tutorials written by the author himself. Additionally, new tutorials for GoDot now appear in the **UCUGA's Commodore Digest** publication.

No more waiting for GoDot!

As stated earlier, CMD marketed GoDot to the North American audience and marketed it with an English printed manual. Now, Arndt Dettke has decided to release the entire GoDot program on his website for free download. Granted, you would not be able to acquire the English manual and unless you have the Go64!/UCUGA issues, you would also miss out on some tutorials. Using GoDot would require a learning curve in order to use it effectively. But, hopefully for the most of the North American audience, using it regularly is the best way in mastering GoDot. In addition to getting GoDot from a users group like LUCKI, you can download GoDot at:

http://members.aol.com/howtogodot/godnews.htm

GoDot, while it can run in stock Commodore systems, is already 'CMD-Aware' and works just fine in a fully-tricked out Commodore setup. Due to heavy emphasis on its statistical and mathematical framework, GoDot benefits tremendously from the SuperCPU running at 20MHz.

All GoDot does is to deal with graphical images and to manipulate them with sheer mathematical precision.

ConGo, a **Windows** program, can also manipulate 4bit files. For those who dabble in **LINUX**, **XnView** can also view 4bit files. The 4bit files can even be viewed by my **dotView** utility within the comforts of the Wheels operating system. While GoDot does not run under **GEOS/Wheels**, its user interface is one of the best in the business, being easy to operate and is intuitive.

The linchpin becomes the Achilles Heel

Up to this point in the article, I have been very positive towards GoDot and it is very well deserved in its praise. However, GoDot, despite its *newness* to the North American audience, has begun to show its age. GoDot is already ten years old and its 4bit technology, the very linchpin of its success is also its biggest Achilles Heel. Consider for a second when GoDot was introduced in Germany ten years ago; The prevailing graphical standard was only 16 colors and the high-end computers sported 256 color displays. GoDot brought image processing to the Commodore 64 in style and in a current fashion. Now, in the modern Internet age, 16 million color graphics are the norm with a 24 bit color depth. If I am to convert a JPEG picture to a 4bit file under GoDot, I lose a staggering amount of 20 bits worth of

Used Palette Controls to set desired colors.
Then used the TileClip module below.

Applied the DeepPress module. (below)

Applied the LaPlacian module for a neon effect. (below)

color information in the conversion process. No amount of error correction and statistical sampling, no matter how good, is enough to overcome such a steep loss without the eye noticing.

I must stress that this really isn't a limitation of GoDot. GoDot is optimally designed for the Commodore 64 and its inherent graphical limitations. The 4bit graphical bitmap format was designed to accommodate the Commodore 64 as it can go up to 16 colors and the resulting bitmap filesize is just 32K bytes. A complete 4bit image bitmap can fit easily within the Commodore 64's onboard memory and still be flexible enough to be seen easily onscreen either in multicolor mode or high resolution mode.

The main thrust of this seemingly simple exercise in using GoDot shows that I haven't created anything from scratch.

The color beast becomes
a three-headed chimera

Moreover, modern computers and related peripherals such as monitors and printers split up color information primarily in three ways. Computers do not know colors, but computers can use the physical rules of nature in creating colors. An artist can mix colors and create new colors in doing so. Color charts have been produced and made specially for artists in mixing colors as to achieve the desired color. The same rules apply to computer generated colors; Computers can mix the Red, Green and Blue beams in a monitor to come up with the desired color. (Modern color printers use a similar scheme called **CYMK**.) This scheme is called **RGB** and carries with it a broad spectrum of colors that a computer can understand and generate. More importantly, the RGB scheme is fully indexed and can be manipulated with equal mathematical force and sheer statistical precision.

Before the first line of code was written for GoDot, the authors collaborated and settled on the actual RGB values which would make up all the values on this 4bit brightness scale. While GoDot would be optimally designed for the Commodore 64, it still retains its freedom by using actual RGB values in its brightness scale. This concept has made graphical conversions between graphics in different platforms, be they Windows, **Amiga** or the Commodore itself all that much easier. Keep in mind that the RGB values underlying the brightness scale is used for image conversions of various graphical formats and is not intended as the foundation behind GoDot in the context of image processing.

GoDot's brightness scale contains sixteen preset colors, according to their luminance. That is fine and is specially optimized for image processing on the Commodore 64. However, color information is indexed according to their brightness and it would be very nice to have them indexed according to this RGB spectrum where such color information is split three ways. The power of image processing is multiplied exponentially when such information can be interpreted and manipulated in three different ways. Color information interpreted in this manner also makes printing and graphical conversions between different computer platforms more easier and more eye-pleasing.

A finer resolution...

Another problem is that GoDot seemingly weds itself to the maximum image size of 320x200 pixels, the very same dimensions of the high resolution screen of the **VIC-II** chip in the Commodore 64. While a 320x200 graphical bitmap is large enough

during the Commodore 64's heyday, modern images sport far greater sizes than that. Again, a conversion of a larger bitmap into the smaller confines of a 320x200 image under GoDot will lead to a loss in image quality, no matter how good the underlying mathematical contortions it underwent, noticeable to the naked eye.

Images is in the eye of the beholder

Despite the graphical limitations of the Commodore 64, GoDot continues to surprise me with its color depth and its image processing abilities maximizing such color information. For example, if you were to process images as a Interlaced (IFLI) image and save it as a EPS file, you would seemingly print out 256 colors, not sixteen! (The PostScript II printers nowadays have finer dot resolution as to fool the eye in displaying more colors; only 136 colors are actually used.) If you were to save this same image as a PCX file for later viewing on your modern computer, you would see up to 136 colors, not sixteen. Granted, its not 16 million colors, but GoDot does a good job in maximizing color information.

More amazingly, GoDot can preprocess images and split up such color information three ways, by using the **Scanntronik** digitizer cartridge and a RGB splitter. A person, **Frank Wagenknecht**, in Germany, captured such images within GoDot using this cartridge and the RGB splitter. In doing so, Frank created three standalone 4bit files, one for each of the Red, Green and Blue color intensities. Arndt then combined the three 4bit files into a single image using **PaintShop Pro** on his modern computer in its full color glory. Such combined images look incredible and carry up to 4096 colors! GoDot can load these '12bit' images via the **4BitRGBd** loader module w/ dithering. Not too bad for a image processor running on a Commodore 64 which only has up to 16 luminance values to play around with? :) Look for more 4096 color images created by GoDot at the following website:

http://members.aol.com/howtogodot/fgaler05.htm and .../fgaler06.htm

North American Commodore audience undergoing a metamorphosis

In a play by **Samuel Beckett**, *Waiting for Godot*, was about a bunch of people waiting for a certain figure named Godot. What was truly remarkable about the play was that it wasn't focused on the Godot person, but was upon the sheer dynamics and change that this bunch of people underwent while waiting for this singular person. You could say that the same thing happened to the North American Commodore audience with respect to the GoDot software. When GoDot was introduced to North America via CMD, it introduced such a paradigm shift from image creation to image processing. We finally began to use GoDot, but we were still conceptually stuck in the ancient ages of image creation and even demanded that Arndt Dettke, GoDot's author, create **mod.PixelEdit** to facilitate our creativity in creating such images.

GoDot is already 10 years old, and its 4bit technology, the very linchpin of its success is also its biggest Achilles Heel.

More importantly, it changed my viewpoint on image processing. No longer did I view the computer as a hostile thing, where I try to master the mouse and/or the joystick in trying to create that perfect screen graphic, frustrating and bottling up my creativity. I begin to look at the computer as my best friend, one of the best tools I can use in processing pre-existing images and unlock my creativity in producing images for the Commodore 64. I began to see the true benefits of image processing; that it marries the power of computers in manipulating the sheer volume of numbers which compose an image bitmap and the power of the human creative soul.

We are still waiting for GoDot, after all!

I also saw the limitations and inherent weaknesses of what is inarguably the first generation software offering which promises to do image processing for the Commodore 64. Image processing software on modern

Bottom Left: The infamous dancing baby finds its way onto the Commodore 64/128.

Left: GoDot, a half-asleep giant, ready to pounce on stray pixels.

computing platforms are now in their tenth (or more) iteration and have far more complex tools and features. I thank Arndt Dettke and **Wolfgang Kling** for their foresight and their ambitious collaboration in bringing forth the very concept of image processing to our Commodore 8-bit line of computers and for their continued support and guidance in the past ten years. I thank them for giving me the tantalizing prospects of which image processing can deliver and what GoDot has delivered so far.

While GoDot has been introduced ten years ago, I have sincere hopes that it is not also the end of image processing as we now know it for the Commodore 8-bit line. In a sense, while we are 'waiting' for this next generation software, we are now figuring out GoDot and understanding image processing concepts as the new paradigm governing graphical bitmaps on our Commodores. I suspect that many of us Commodore users, especially now that GoDot is freely available for download, will undergo a similar sort of metamorphosis envisioned by Samuel Beckett in that classic play.

Peering into the crystal ball...

I have glimpsed a little bit into the inner workings of GoDot, with many thanks to Arndt Dettke lending me some of the source code for GoDot. I am impressed with its conceptual framework using mathematics and statistics in processing image bitmaps. More importantly, Arndt Dettke is still working on GoDot. He is updating his conceptual framework for image manipulation by using the Commodore REU.

Before the first line of GoDot was written, the authors collaborated on actual RGB values comprising the 4bit brightness scale, giving it a degree of independence in conversions.

With GoDot's ability to use the REU effectively, the user can work on more than one image, use an image as a mask, hold graphics clips, have a powerful undo function, easily use GoDot modules w/o disk access, and much more. Arndt is now busily working on making a JPEG loader module for GoDot, and have an IFLI viewer function as well. All of the latest programming advances in GoDot are made possible by the REU. Time will tell if the SuperCPU, will too, receive the GoDot treatment and have a native version with even more powerful image processing capabilities.

The future awaits us and in the meantime, enjoy image processing on your Commodore 64 using GoDot. Let creativity guide your usage of GoDot.

(*Continued from Page 1*)

Laser Lovers
Wrong Is Write 81
UCUGA Commodore Digest
Code by **Maurice Randall** and **Joe Buckley**, respectively. Promoted by **K. Dale Sidebottom**. If you want the finest in *Commodore GEOS printing*, check it out!
www.luckyclub.net

ConGo
Created and maintained by **Matthias Matting**.
www.editorix.org/congo/

XnView
Created and maintained by **Pierre Gougelet**.
www.xnview.com

dotView
Created and maintained by **Todd Elliott**
http://www.geocities.com/eyethian2000/dotview.html

VICE
The ultimate C64/128 emulator.
viceteam.bei.t-online.de

Star Commander
Maintained by **Joe Forster**. A cool interface for maintaining CBM/GEOS files on modern platforms.
sta.c64.org

CC65 Programming Suite
Maintained by **Ullrich Bassewitz**. GEOS Support by **Maciej Witkowiak**.
www.cc65.org

Eyeth wears a catcher's mask to ward off color clashes.