The NGT30 is an easy to use, 5v compatible propeller graphics device. It communicates simply with UART communication.

It will help you make your project by featuring:

- An Arduino shield-compatible interface
- Supports 5v and 3.3v interface
- Does basic 3D graphics efficiently
- Pins used on Arduino can be configured
- Quick Sprite definition and display
- Easy to modify the NGT30's firmware
- VGA without costing tons of RAM, program space, and CPU time

Please let the table of contents help you find what you need to know.

Table of Contents

Conventions

- Command arguments sent to the NGT30 are in Big Endian format!
- Data loaded into VRAM is in Little Endian format!
- A byte is 8 bits
- A word is 2 bytes
- A long is 4 bytes

NMT Graphics Card NGT30 Overview

What can it do?

- The NGT30 produces VGA video output
- Draws sprites
- Can do rainbow colors in foreground or background across tiles
- The firmware is open-source, so if you like you may modify your NGT30’s firmware
- Can be used as an arduino shield, or hacked in various ways

How does it work?

The NGT30 uses the **parallax propeller,** an 8-core microcontroller, clocked at 80 MHz. It is excellent for producing video signals, which is why it was chosen to make the NGT30. It runs at 3.3v, so it cannot directly connect to a 5v arduino, Z80 UART, or other 5v device, so the NGT30 uses a **voltage level shifter.** It **uses multiple cores** to generate video. It communicates with the host device with **115200 bps UART.** If you modify the firmware, or otherwise reprogram the NGT30, the way that it communicates can change.

What sets NGT30 apart from other graphics shields?

- 3D Graphics. In Color.
- Compatible with more host devices than just an arduino, anything capable of communicating on 115200 bps UART at 3.3v-5v will be compatible.
- Can draw rainbow text and sprites
- Supports the quick repeated drawing of sprites, as opposed to pixel-by-pixel draws
- Advanced arduino library for graphics and creating sprites at runtime
- Easily reprogrammed with a Prop Plug, as opposed to FPGA or CPLD versions
- Does not have to be used as an arduino shield or with a host device, but can be used as a propeller development platform (pins 1-8 are broken out and two video outputs are present)

In short, it is an **easy to use**, **VGA output**, **expandable** and **reprogrammable** propeller graphics device.

How To Use The NGT30 With Arduino

Here begins directions on how to get off the ground with your new NGT30 board.

Connecting the NGT30 to your Arduino

A note on communication voltages:

- Since the serial output from the NGT30 is always 3.3v, regardless of what TX pin it's from, **3.3v-compatible devices will never have problems connecting to the NGT30 from voltage incompatibilities. This guarantees compatibility with 3.3v arduino boards.**

Stack the NGT30 onto the Arduino, with the VGA connector the opposite way of the arduino's USB and Barrel Jack connectors.

The NGT30 uses Arduino pins 6 and 9, 6 being the Arduino's RX and 9 being the TX. These pins are used by the Arduino Library, but it is important not to connect anything else to the pins.

For interface with the NGT30 via arduino boards the NMT_GFX Arduino library can be used. It is best to have the library. Download it from GitHub here.

To initialize the NGT30 it the library must be set up first:

```
#include <NMT_GFX.h>
NMT_GFX ngt;
```

Easily done, simply includes NMT_GFX and creates an object to use. Now to actually initialize the NGT30:

```
ngt.begin(11,10);
```

ngt.begin properly resets the NGT30 so it is ready to receive commands. Now the NGT30 is initialized.

Colors and the NGT30

It is important to understand how colors work specifically for the NGT30. Please read this section to understand.

The NGT30's bitmap is divided into tiles. The default NGT30 firmware has 16 x tiles and 12 y tiles. Each tile has 16 columns and 16 rows. Each tile also has only 4 colors. These colors must be selected from one of 64 'color slots'. To make this concept easier to understand, here is an example:

Tile X is currently using color slot zero. There is text on the tile with the foreground using color 2 of the color slot's 4 colors. By changing color slot zero's color 2, that text's color changes for all tiles using that color slot, including tile X. If tile X's color slot is changed to slot 5 for instance, all the pixels in tile X will adopt their corresponding colors in slot 5.

Although this could seem a disadvantage, it can be useful for making a sprite change color as it moves or making rainbow text. It also allows the background of certain tiles to change, or text to change color at certain boundaries. There are both pros and cons to this approach to displaying colors.

List of arduino library functions

The NMT_GFX Arduino library is available here.

```
NMT_GFX::NMT_GFX();
```
Starts the NMT_GFX library with NGT20-compatible configuration.

`NMT_GFX::NMT_GFX(rx, tx);`
Starts the NMT_GFX library with the specified RX and TX pins.

`void NMT_GFX::print(x);`
Print character or string x at the cursor and advance the cursor once.

`void NMT_GFX::println(x);`
Print the string or char x onto the screen and advance the cursor to the end of the string, then go to the next line.

`void NMT_GFX::write_at(q, int x, int y);`
Print the string or char q onto the screen where the upper left of the string is placed at (x, y). Does not affect cursor position.

`void NMT_GFX::set_cursor_pos(int x, int y);`
Set cursor position to (x, y). Note that unlike other functions, x and y are measured in characters, which are 6x13 pixels each.

`void NMT_GFX::line(int x1, int y1, int x2, int y2);`
Draw line from (x1, y1) to (x2, y2). (x2, y2) is new endpoint.

`void NMT_GFX::box(int x1, int y1, int x2, int y2);`
Make a square between (x1, y1) and (x2, y2)

`void NMT_GFX::fill_box(int x1, int y1, int x2, int y2);`
Make a solid square between (x1, y1) and (x2, y2)

`void NMT_GFX::fast(int x, int y);`
Makes a line between the endpoint of the last line or the last pixel, whichever occured last, and (x, y). New endpoint is (x, y)

`void NMT_GFX::clear();`
Fill every tile with color zero in its color slot.

`void NMT_GFX::fill(byte color);`
Fill every tile with color in its color slot.

`void NMT_GFX::set_color(byte color);`
Make next draws use color.

`void NMT_GFX::pixel(int x, int y);`
Draw pixel at (x, y)

`byte NMT_GFX::x_tiles();`
Returns number of x tiles. Multiply this by 16 to get number of columns.

`byte NMT_GFX::y_tiles();`
Returns number of y tiles. Multiply this by 16 to get number of rows.

`char* NMT_GFX::get_card_ver();`
Returns string for device model. For an NGT30 it should be "NGT30". Newer or older versions may say something different, like "NGT20".

`void NMT_GFX::block_color(byte slot, byte color);`
Set color (slot>>6) in slot (slot&63) to the argument 'color'.

`void NMT_GFX::tile_color(int tile, byte slot);`
Make a tile use a specific color slot.

`byte NMT_GFX::make_color(byte r, byte g, byte b);`
Returns a color for use in `NMT_GFX::block_color()` using RGB color input.

`uint16_t add_line(long x1,long y1,long z1,long x2,long y2,long z2,uint8_t color);`
Adds a line to the NGT30's memory to be rendered. Returns the line's ID number, for removal later if necessary.

`void del_line(uint16_t id);`
Deletes a line using the line's ID number.

`void render_3d();`
Renders one 3D frame, without clearing the screen first.

`void frame_3d();`
Renders one 3D frame, with clearing the screen first.
 * Note: It is not recommended to call this too fast, otherwise the screen may clear faster then rendering, thus making the graphics difficult to see onscreen.

`void translatef(long x,long y, long z);`
Moves the camera to the coordinates (x,y,z). Does not render a new frame!

`void rotatef(uint16_t yaw, uint16_t pitch);`
Sets the camera yaw and pitch. Does not render a new frame!

There are code examples for many of these in the NMT_GFX library.

Creating, loading, and displaying a sprite definition

All examples here are written in C++ for the Arduino. They use the NMT_GFX Arduino library available here.

All the code I present here will be in the Arduino function “setup”, except for variable declarations.

This example uses NMT_GFX’s Sprite class. It is a simple and easy to use class that only requires the definition of the sprite size and center, some RAM to use, and some initialization. First some memory must be allocated for the Sprite:

```
Byte image[52];
```

This is enough space for an 8x24 sprite. 4 bytes are used for parameters and 48 bytes times 4 pixels/byte = 192 pixels. X length has to be a multiple of eight in RAM, so 192/8 = 24 y pixels. Ok, here is where the sprite is defined:

```
Sprite sprite;
```

No fancy constructor, just the definition. It is initialized by these lines:

```
// tell sprite where to store its data
sprite.binary_image=(byte*)image;
// tell sprite how big it is
sprite.set_size(8,24);
// set the center at 4,4 pixels
sprite.set_center(4,4);
```

The sprite is told the location of its memory, its size, and its center. Now it is ready to be written.

The writes are done with function calls to Sprite::fill() and Sprite::pixel()

```
// fill sprite
sprite.fill(2);
// draw some pixels
sprite.pixel(3,3,0);
... And more writes ...
```

sprite.fill(2) makes all pixels in the sprite color 2 of whichever color slot that pixel occupies. The color slot occupied is the pixel's tile's color slot. To understand how these colors work see Colors and the NGT30.

sprite.pixel(3, 3, 0) makes the pixel at (3, 3) color 0 of its color slot.

Once all writes are done on the sprite it MUST be uploaded to the NGT30:

```
sprite.upload();
```

Easy. Now we can put it across the screen a ton of times:

```
// Display sprite at 50,60 at a 0 degree angle
sprite.display(50,60,0);
sprite.display(20,60,0);
sprite.display(60,60,0);
```

These lines display the same image 3 times, each time minimal exchange happens between the NGT30 and the Arduino, as opposed to slowly and painfully writing raw pixels and boxes to the screen, in a repetitive inefficient process.

sprite.display()'s 3rd argument specifies the orientation of the sprite. 0 is 0 degrees, 1 is 90 degrees, 2 is upside down, and 3 is 270 degrees.

Now things get even cooler though. The array of bytes given to the Sprite is an image and can be saved somewhere and reloaded later. To reload a Sprite from a raw memory/binary image simply do this:

```
sprite.binary_image=your_saved_image_as_byte_array
```

Now you can save a sprite in eeprom, on an SD card, and more, then reload it later.

Rendering in 3D

All examples here are written in C++ for the Arduino. They use the NMT_GFX Arduino library available here, and also the PS2Keyboard library available here.

The NGT30 can render frames consisting of 3D lines. The NGT30 saves the lines and camera transforms so that a single byte command can draw a new frame.

To explain this, we will create a simple 3D world with only a cube and a camera in Arduino C++.

First, the Keyboard and Graphics libraries must be loaded.

```
#include <NMT_GFX.h>
#include <PS2Keyboard.h>
NMT_GFX ngt;
PS2Keyboard keyboard;
```

Now we have our graphics object, `ngt`, and our keyboard object `keyboard`.

In our `setup` method we must actually initialize these. The calls are important because the numbers tell the libraries which pins to use.

```
keyboard.begin(4, 2);
ngt.begin(11,10);
```

The keyboards IRQ pin is 2 and dat pin is 4 on the NGT30. The NGT30 default pins are 11 as RX and 10 as TX. Since the NMT_GFX library supports any pins compatible with a SoftwareSerial port, it needs to know which ones to use.

Now we can tell the NGT30 the coordinates for our cube:

```
ngt.add_line(10,10,10,30,10,10,1); // that last argument that
ngt.add_line(30,10,10,30,30,10,1); // stays '1' is the color
ngt.add_line(30,30,10,30,30,30,1); // of the lines.  Default
ngt.add_line(30,30,30,10,30,30,1); // color scheme will make
ngt.add_line(10,30,30,10,10,30,1); // the lines white.
ngt.add_line(10,10,30,10,10,10,1);
ngt.add_line(10,10,30,30,10,30,1);
ngt.add_line(30,10,10,30,10,30,1);
ngt.add_line(10,10,10,10,30,10,1);
ngt.add_line(30,10,30,30,30,30,1);
ngt.add_line(10,30,10,30,30,10,1);
ngt.add_line(10,30,10,10,30,30,1);
```

The cube has 12 edges to define, each with different coordinates. That code tells the NGT30 about all of the edges and their coordinates.

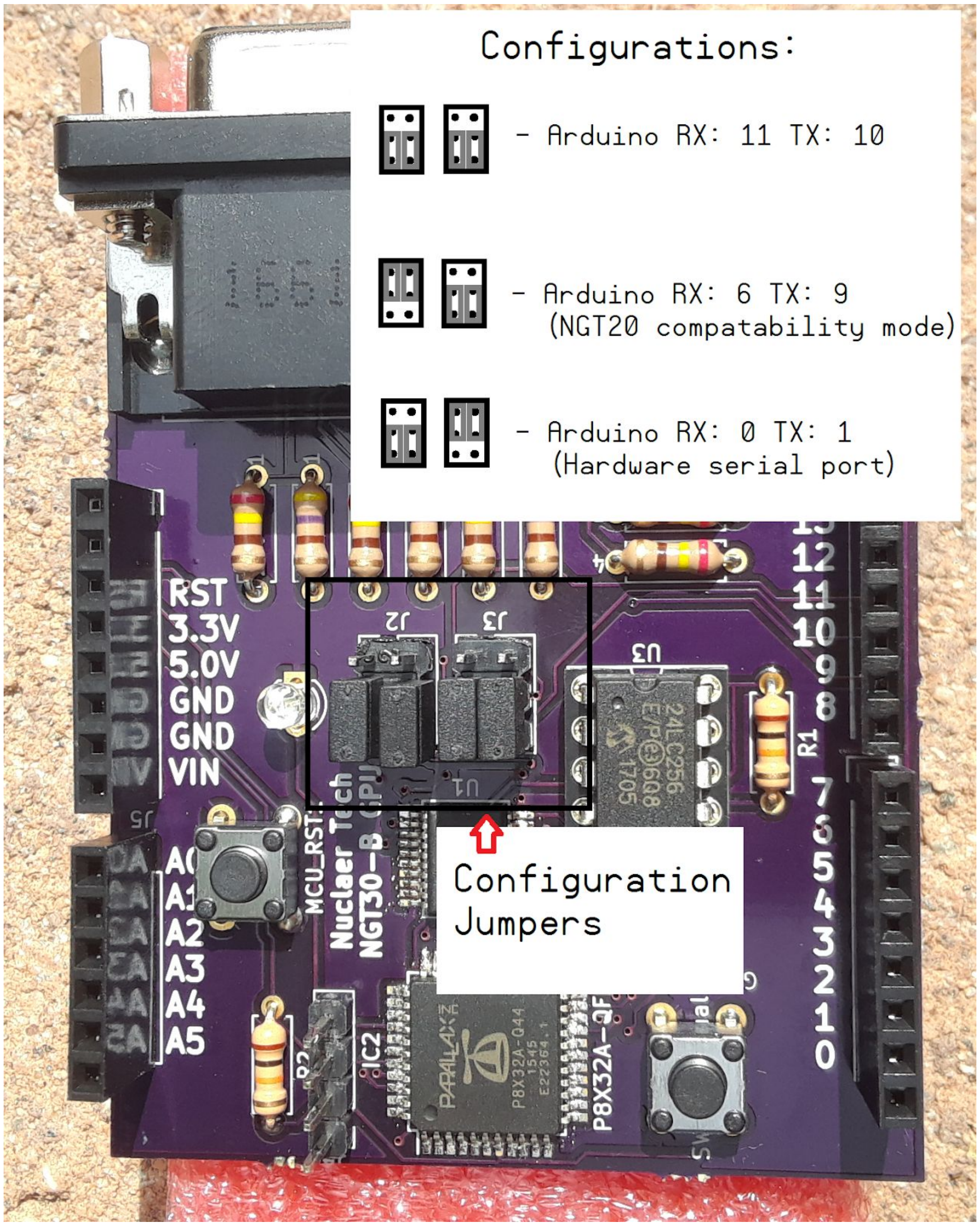
Now we can make a game loop:

```
double x=0,z=0; // keep track of the coordinates and rotation
int rot=0;
while(true){
  int32_t time=millis();
  if (keyboard.available()) {
    // read the next key
    char c = keyboard.read();
    if (c == PS2_LEFTARROW) {
      rot+=3;
    } else if (c == PS2_RIGHTARROW) {
      rot-=3;
    } else if (c == PS2_UPARROW) {
      x-=cos(rot*PI/180);             // use trig for the movement.
      z+=sin(rot*PI/180);             // this is so the camera moves
    } else if (c == PS2_DOWNARROW) {  // in the direction it looks.
      x+=cos(rot*PI/180);
      z-=sin(rot*PI/180);
    }
    ngt.rotatef(rot,0);                          // rotate the 3D camera
    ngt.translatef((int32_t)x,0,(int32_t)z);  // move the 3D camera
  }
  ngt.frame_3d();       // frame_3d both clears the screen and
renders.
  delay(max(1,(16+time)-(int32_t)millis())); // cap the FPS at 60
}
```

This game loop allows the use of a keyboard to move in a 3D world. When a person presses the keyboard, the arduino calculates the new position of the camera and sends it to the NGT30. The game loop also renders 60 or less times a second, so that the NGT30 is never clearing the screen for more than half the time (thus making the video output dark and hard to see).

Configuring pin connections.

The NGT30 can have its pins connected to different Arduino pins with jumpers on the board to avoid pin conflicts.



Configurations:

- Arduino RX: 11 TX: 10

- Arduino RX: 6 TX: 9
  (NGT20 compatability mode)

- Arduino RX: 0 TX: 1
  (Hardware serial port)

Troubleshooting

NGT30 won't display anything. TV/Monitor gets signal.

The NGT30 may be configured to use different pins than the Arduino is expecting, or the code for the Arduino is connecting on the wrong RX/TX pins. Typically, the NGT30 has RX and TX pins configured as 11 and 10 respectively, however this can be adjusted with the jumper wires on the NGT30.

NGT30 was working, then green light turned solid and it stopped responding.

This will happen if an invalid sprite is displayed. Simply hard reset the NGT30 to set to original state.

Colors and the NGT30

Please read this for proper understanding of how colors work for the NGT30.

The NGT30's bitmap is divided into tiles. The default NGT30 firmware has 16 x tiles and 12 y tiles. Each tile has 16 columns and 16 rows. Each tile also has only 4 colors. These colors must be selected from one of 64 'color slots'. To make this concept easier to understand, here is an example:

Tile X is currently using color slot zero. There is text on the tile with the foreground using color 2 of the color slot's 4 colors. By changing color slot zero's color 2, that text's color changes for all tiles using that color slot, including tile X. If tile X's color slot is changed to slot 5 for instance, all the pixels in tile X will adopt their corresponding colors in slot 5.

Although this could seem a disadvantage, it can be useful for making a sprite change color as it moves or making rainbow text. It also allows the background of certain tiles to change, or text to change color at certain boundaries. There are both pros and cons to this approach to displaying colors.

Technical Specifications

Communication protocol of the NGT30

The NGT30 accepts commands on the UART port and interprets them. It then returns a result on the UART. Some commands take arguments to give the NGT30 more information about what it is being told to do.
Note that:

- All commands are sent in sequences of bytes
- String arguments are NOT zero terminated, but CR-terminated.
- Any unlisted command IDs have no effect on the NGT30. For a more advanced device, the effect will be different and that device's datasheet should be referenced.

To run a command on the NGT30 first a the command ID must be sent, then its arguments. The arguments are in the same order as specified in the table under "Commands At A Glance".

Argument Data Types

Different arguments are different data types.
Data types include:

- BYTE
- WORD (2 bytes)
- STRING (any number of bytes, CR-terminated)

Any command with arguments will accept more data after the sending of the command ID. For each argument data must be supplied, the length of each argument depending on data type. Refer to the bulleted list above, and remember to CR-terminate all STRINGs.

Refer to the table below for the arguments of each command.

Commands At A Glance

Command ID	Command Description	Arguments
13	Newline	None
32	Get device type (CR-terminated)	None
48	Reset NGT30	None
49	Clear Screen	None
50	Set Cursor Pos	BYTE x, BYTE y

51	Write string	STRING str, BYTE 0x0D
52	Draw line	WORD x1, WORD y1, WORD x2, WORD y2
53	Fill Screen with color	None
54	Fill box	WORD x1, WORD y1, WORD x2, WORD y2
55	Write string at position	WORD x, WORD y, STRING str, BYTE 0x0D
56	Write pixel	WORD x, WORD y
57	Set Color	BYTE color
61	Change color slot	BYTE slot, BYTE color
62	Draw sprite	WORD x, WORD y, BYTE rotation, WORD address
63	Write RAM	WORD address, BYTE data
64	Get output type	None
66	Set tile color	WORD tile, BYTE slot
67	Fast line	WORD x, WORD y
100	Render 3D Frame	None
101	Get Graphics Memory Length	None
102	Set camera position	LONG x, LONG y, LONG z
103	Set camera rotation	WORD yaw, WORD pitch
104	Set 3D line	BYTE index, WORD pointer
105	Disable 3D line	BYTE index

Commands - In Depth Descriptions

Please refer to the table above for information on command arguments.

Command 13 : New Line

Will advance the cursor downward 1 character. If the cursor leaves the screen then the entire screen is scrolled up. This affects all pixels on the screen, so be careful or use this to your advantage.

Command 32 : Get Device Type

Will send a null terminated string indicating the device type. For the NGT30 it would literally be "NGT30", then a NULL.

Command 48 : Reset

Will completely reboot the NGT30, all pixels, sprites, and colors will be lost. Equivalent to pressing the reset button, except Command 48 requires no mechanical action.

Command 49 : Clear screen

Sets cursor to top left and makes every pixel use it's tile's zeroth color. Note that this means pixels in different tiles may have different colors due to the tiles having a modified tile colors. Use Command 61 to change a color set's colors and Command 66 to change a tile's color set.

Command 50 : Set Cursor Pos

Set cursor position for Command 51. Measured in characters. Each character is 6 pixels wide and 13 pixels tall.

Command 51 : Write string at cursor position

Writes a carriage-return terminated string on the screen and advances the cursor to after the string's end. If the string goes past the end of the screen it loops back on the other side. If it continues below the screen's lower edge then the screen scrolls up, including all pixels, and the string continues writing. BE SURE THE STRING IS CR-TERMINATED!

Command 52 : Draw Line

Draws a line from (x1, y1) to (x2, y2).

Command 53 : Fill Screen with color

Fill screen with color. Note that color is NOT an RGB color, but a number from 0-3 referring to tile color. To change the color slot of a tile or colors in a color slot use commands 61 and 66.

Command 54 : Fill box

Fill box with opposite edges at (x1,y1) and (x2, y2)

Command 55 : Write string at position

Writes string at (x, y)
Does not change cursor position. X and y are in pixels, not characters. Also note that the justification of the text places the upper left of the string at (x, y).

Command 56 : Write Pixel

Draws a pixel at (x, y)

Command 57 : Set Color

Set color for next draw/text commands.   Note that color is NOT an RGB color, but a number from 0-3 referring to tile color. To change the color slot of a tile or colors in a color slot use commands 61 and 66. Does not apply to command 49 (Clear Screen)

Command 61 : Change color slot

There are 64 color slots. The lower 6 bits of the slot argument define which color slot. The upper 2 bits select one of 4 of the 1-byte color definitions for that slot.

Command 62 :Draw sprite

Draws a sprite at (x, y). Rotation is 0-3, where 0 is no rotation, 1 is 90 degrees, 2 is 180, and 3 is 270. Sprite is the offset of a sprite definition written to the NGT30's RAM. For instructions on creating a sprite definition, loading it, and displaying it refer here.

Command 63 : Write RAM

The NGT30 has 3 KiB of RAM that can be written to store sprites. You could store about 255 8x4 pixel sprites with this space. Command 63 writes one byte of this memory for loading sprites to the NGT30. For instructions on creating a sprite definition, loading it, and displaying it refer here.

Command 64 : Get output type

This command returns data about your NGT30 depending on its firmware. The command sends back 3 bytes of data. The first byte is the number of x tiles. Every X tile has 16 columns. The second byte is the number of y tiles. Every Y tile has 16 rows.

The last byte should be 'D' if the NGT30 has the default firmware. If the firmware does not support NTSC output the char will be 'V'. If it only supports NTSC the char will be 'N'.

Command 66 : Set tile color

Sets specified tile to use specified color slot.

Command 67 : Fast line

Excellent for drawing lines fast - takes about half the time of a normal line draw.

If a line was drawn last then it's endpoint becomes the fast line's starting point. If it was a pixel then that pixel becomes the starting point. The fast line's endpoint is (x, y).

Command 100 : Render One 3D Frame

Renders one 3D frame.

Command 101 : Get Graphics Memory Length

Upon executing this command the NGT30 will send back a word which is the number of kilobytes (1024 bytes, not 1000 bytes) that command 63 has available to write to.
This command can be used to check if it is safe to write to an address of VRAM.