

**Chunky**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> Chunky	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY		January 19, 2025
<i>SIGNATURE</i>		

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>Chunky</b>	<b>1</b>
1.1	Chunky V1.00 . . . . .	1
1.2	allocatchunkybuffer . . . . .	1
1.3	chunkyblit . . . . .	2
1.4	chunkyblock . . . . .	2
1.5	chunkycls . . . . .	2
1.6	chunkyid . . . . .	3
1.7	chunkyplot . . . . .	3
1.8	chunkytoplanar . . . . .	3
1.9	freechunkybuffer . . . . .	3
1.10	initchunky . . . . .	4
1.11	usechunkybuffer . . . . .	4

---

# Chapter 1

## Chunky

### 1.1 Chunky V1.00

PureBasic - Chunky library V1.00

'Chunky' is a method to store the display data, a bit like the planars. But, this method works at the opposed way as the planar. Each pixels are represented by one byte in the memory, so you can have only 256 colours screens (as one byte is 8 bits, or 8 planes if you prefer). The Amiga can't display such graphics data and a conversion must occur. It's the well know ChunkyToPlanar routine. Working on chunky is faster for many operations but as only the cpu is involved (and no Amiga custom chips) you will need a fast computer to have a descent speed.

Commands summary:

```
AllocateChunkyBuffer
ChunkyBlit
ChunkyBlock
ChunkyCls
ChunkyID
ChunkyPlot
ChunkyToPlanar
FreeChunkyBuffer
InitChunky
UseChunkyBuffer
```

### 1.2 allocatetchunkybuffer

SYNTAX

```
ChunkyID.l = AllocateChunkyBuffer(#ChunkyBuffer, Width, Height)
```

FUNCTION

It will allocate a ChunkyBuffer. A so called ChunkyBuffer is a memory area which is the reflect of a 256 colours screen of the given dimension. As the Amiga isn't able to display this kind of graphics, you will need a conversion algorith, called a Chunky2Planar which will

---

display the content of the chunky buffer on the standard Amiga planar graphics. In fact the chunky is the opposed method to the planars. The main advantage of the chunky is you can move big blocks of memory (like sprites, 3D engines) very quickly. The drawback is you can't use the Amiga custom chips (Blitter, Copper...) to accelerate the things, so you will need a fast computer (030 or better) to achieve a descent frame rate. The chunky format is used on graphics cards too, so you can directly copy the chunky buffer to the video ram.

The returned ChunkyID is the memory location of the ChunkyBuffer. If its NULL, the chunky buffer isn't allocated, so don't perform any operations on it.

The ChunkyBuffer will be allocated at #ANY\_MEMORY place (in FastRam if you have one, or else in ChipRam)

### 1.3 chunkyblit

#### SYNTAX

ChunkyBlit(ShapeWidth, ShapeHeight, \*ShapeAdress, X, Y )

#### FUNCTION

Blit the specified chunky shape to the given Chunky buffer. This function uses colour 0 as transparent. It's a highly-optimized function. The shape can be of any size. This function isn't clipped, so be sure to be INSIDE the chunky buffer when you perform your blit.

### 1.4 chunkyblock

#### SYNTAX

ChunkyBlock(ShapeWidth, ShapeHeight, \*ShapeAdress, X, Y)

#### FUNCTION

This function is about 5 times faster than the ChunkyBlit function but has some limitations:

- \* The shape width must be a multiple of 4.
- \* There are no transparent colours.

This function isn't clipped so be sure to Blit INSIDE the buffer.

NOTE: This is the fastest way to Blit a shape inside a chunky buffer. This function has been specially optimized for speed.

### 1.5 chunkycls

#### SYNTAX

ChunkyCls(Colour)

---

## STATEMENT

Fills the current chunky buffer with the specified colour.

## 1.6 chunkyid

## SYNTAX

ChunkyID.l = ChunkyID()

## FUNCTION

Returns the memory location of the current ChunkyBuffer.

## 1.7 chunkyplot

## SYNTAX

ChunkyPlot(X, Y, Colour)

## STATEMENT

Draws a plot at coordinate (X,Y) with the specified colour.

## 1.8 chunkytoplanar

## SYNTAX

ChunkyToPlanar(BitMapID, OffsetY, Height)

## FUNCTION

This will convert the current chunky buffer to the given BitMap. This function is very restrictive and you must follow carefully theses rules:

- The Width must be equal to 320. No other width are supported
- The BitMap and the ChunkyBuffer must have the same size
- The BitMap must be allocated with AllocateLinearBitMap()

Note: the height is not limited.

This function is of course very fast and OS friendly. You can use it quietly in your applications/programs.

## 1.9 freechunkybuffer

## SYNTAX

FreeChunkyBuffer(#ChunkyBuffer)

## STATEMENT

Free the given ChunkyBuffer (and its memory).

---

## 1.10 initchunky

### SYNTAX

```
result.l = InitChunky(#NumChunkyBufferMax)
```

### FUNCTION

Initializes the Chunky environment for later use. You must put this function at the top of your source code if you want to use the Chunky commands. You can test the result to see if the Chunky environment is correctly initialized.

#NumChunkyBufferMax : Maximum number of ChunkyBuffers to handle.

## 1.11 usechunkybuffer

### SYNTAX

```
UseChunkyBuffer(#ChunkyBuffer)
```

### STATEMENT

Set the given #ChunkyBuffer to current ChunkyBuffer.