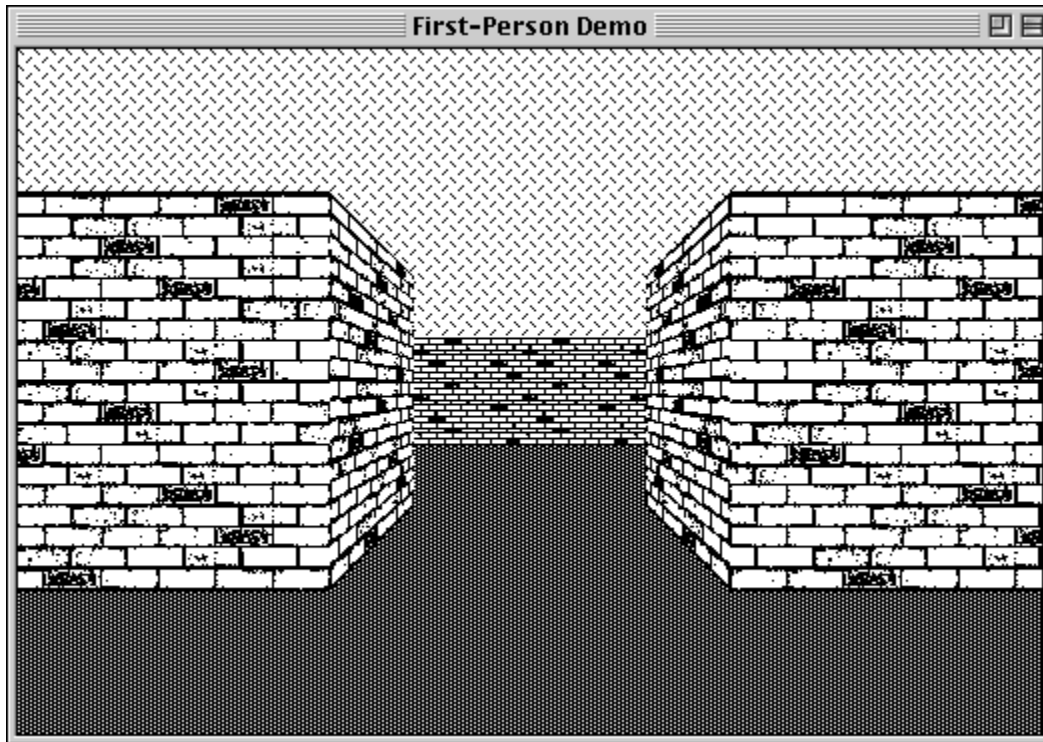


HyperCard-based First Person Game Engine



Chris Tully | Jan 2023

c.a.s.tully@gmail.com

Download from my Portfolio on Google Drive,

<https://goo.gl/nG4kMp>



www.hackaday.io/cast4

Or from Hackaday IO.

Contents:

[What This Is](#)

[Why Make Such a Thing?](#)

[Can This Run on Original Hardware?](#)

[How Could Speed be Improved?](#)

[System Requirements](#)

[Troubleshooting](#)

[How Do I Play?](#)

[How Does It Work?](#)

[The Visibility Problem](#)

[Unions](#)

[Special Cases](#)

[Under the Hood](#)

[Glossary](#)

[HC and Macintosh Specific Terms and Acronyms](#)

[Terms Specific to this Stack](#)

[Stack Structure](#)

[Global Variables](#)

[Stack Script](#)

[Backgrounds](#)

[bg1](#)

[bg_levels](#)

[Cards](#)

[dev](#)

[game](#)

[Other Misc Backgrounds and Cards](#)

[Calculating Reference Data](#)

[View Perspectives](#)

[Ideal Characteristics of a Perspective Equation](#)

[About Shared Textures](#)

[How to Add Additional Textures](#)

[Square "Regular" Textures](#)

[Trapezoid "Side" Textures](#)

[On the Vintage Mac Side](#)

[What Happens If I Do Things Differently?](#)

[How to Add & Edit Levels](#)

[Make a New Level](#)

[Edit an Existing Level](#)

[Current Limitations](#)

[Future Plans](#)

[Discoveries & Trivia](#)

[Acknowledgements & References](#)

[Textures:](#)

What This Is

This is an educational (i.e. slow) tech demo / proof of concept for a first person game engine, built within Hypercard 2.4.

This is not a complete game, yet.
No NPCs, items, or weapons have been implemented yet.

It features 4 small rooms to walk around in (easy enough to change/edit these rooms and add more).

The main achievement of this release is just the functional first-person perspective being drawn to screen.

Expect ugly placeholder textures in 8-bit colour, and dithered black & white.
It currently takes 1-2 seconds for the view to update for every step the player takes.

Why Make Such a Thing?

I like the design and feel of the old Classic Macintosh GUI (resolution of 512x342, black & white). It's clean and simple, but can achieve much with the limited controls it offers.

Some amazing games were made for the platform in it's heyday, and many were very innovative, making the most of the unique and exclusive features that the Macintosh offered, over the competition.

However, many of those games were made before the FPS genre (and gaming as a whole) rapidly evolved through the 1990's.

It would be nice to have an FPS game for System 6 era machines with a few more modern quality of life features and standardised conventions, while also adhering to period Macintosh design rules, and aesthetics.

Being a vintage Mac enthusiast, I eventually got around to giving HyperCard a serious go, working through tutorials, and seeing what all the fuss was about. About the same time, I was playing through Doom RPG (developed in mid 2000's for pre-"smart" mobile phones), and thought that it could just about be recreated with the tools that HC offered.

After 12 months of working on it in my spare time, here's the result.

The main feature of this release is just getting the first-person perspective drawing on screen. The long term goal, is to have something approximating Doom RPG, including it's feature set of:

- different classes of monsters (with simple path-finding)
- items / ammo / health system
- weapons / damage + melee
- interactable terminals and NPCs with text dialog
- locked doors + keys
- environmental hazards (fires to be put out, debris to be cleared)

- a simple shop / trade & inventory system
- different floor types (e.g. lava)
- scripted events (moving NPCs, destructible walls & map layout changes)
- palette based HUD elements
- transparent impassable walls (windows & bars)
- level map / overview (a rogue-like ascii point of view wouldn't take much to complete)
- Hidden doors, secret areas & easter eggs!

Maybe one day, even multiplayer could be implemented over AppleTalk. The technology exists!

Because this is developed in HyperCard, with scripts written in the 35+ year old interpretive language HyperTalk, performance is not great. But one advantage of this, is that because HyperTalk syntax is so close to natural language, it is easy for non-programmers to read through, learn, and make their own changes. Please do!

That fact that I was drawn to create this project in HC says something about it's enduring legacy.

Can This Run on Original Hardware?

Technically Yes, practically No.

This project has been developed with Basilisk II & MinivMac, and performs best within these emulators.

All work completed so far has been done within the confines of Hypercard 2.4 (circa 1998) which does officially support machines as far back as the 1986 Macintosh Plus.

Running on System 6.0.8 in MinivMac emulating a Mac Plus with 4MB of RAM at "All Out" speed, still takes several seconds to update the view for each step the player takes.

I would have no clue how long the scripts in this stack would take to compute on the venerable 68000 CPU.

It is best enjoyed in an emulator, with the horsepower of a modern machine behind it.

How Could Speed be Improved?

No doubt other HC aficionados could optimise the scripts much further.

HC also offers the capability to compile a stack into a stand-alone application. This could potentially offer some modest performance gains, as long as some critical functionality of HC is not removed from the output.

Writing the project in C would help with speed, but I'm not at that level yet. I love programming, but would still describe my skills as "humble", I'm just figuring it out as I go along, for this project. Its been fun though.

System Requirements

This demo requires:

- HyperCard 2.4.1 minimum.
- System 6.0.8 to Mac OS 9.2
- 4MB RAM or higher
- Screen resolution 512x342 or higher.

Additional tools are required if you want to customise the stack with your own textures. See the "[How to Add Additional Textures](#)" chapter for more info.

It is recommended to run this demo on either the Basilisk II or MinivMac emulators. It may run on authentic vintage hardware, but at such a slow speed as to not be playable.

The following Resolution and Colour combinations are available:

	Colour	B&W
512x342 Best for B&W All-in-One Macintoshes.		
512x384 Best for the Macintosh Colour Classic.		X
640x480 Good for Mac II and most other colour capable models into the 1990's.		X

The B&W texture set currently only supports the smallest resolution mode.

Troubleshooting

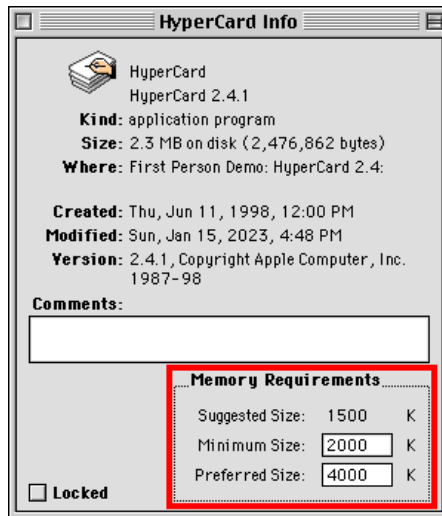
It may surprise you to learn that the Operating Systems for these vintage computers, were not the most stable, and could (*only very rarely*) crash, and potentially corrupt any files in use at the time of failure.

For this reason, it is recommended to keep a back copy of your OS Boot Disk and "First Person Demo" files. This way, if your machine should crash and corrupt your files, you can simply delete them, and make a fresh copy based on your backups.

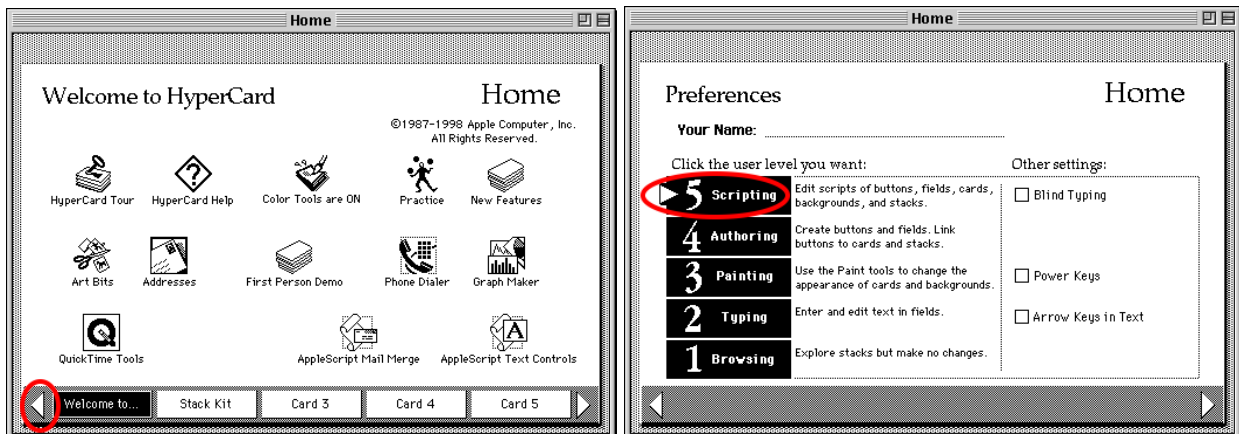
If the stack is not working correctly for you, please ensure that:

- Your HyperCard 2.4.1 application must be set to a higher "Memory Allocation". Single-click on your Hypercard app, and press Cmd + I to open the "Get Info" window.

Edit the Memory Requirements section at the bottom of the window.
 Change the Minimum Size to 2000KB and the Preferred Size to 4000KB
 If this is not done, not all colour textures may be drawn on your screen during gameplay.



- Your “User Level” in HC must also be set to Level 5: Scripting. This can be changed in your Home Stack, on the “User Preferences” card. (Click on the arrow on the bottom left corner of your Home Card).



- If not all Black & White walls are drawing on screen, try the “Recreate B&W Btms” option from the “Developer” menu. The process may take several seconds to complete (in an emulator), during which the screen will be blanked. When the process is complete, press a movement key, and hopefully the missing regions will be drawn on screen correctly.
- If all else fails, delete your copy of the “First Person Demo.dsk” file, and use a fresh copy.

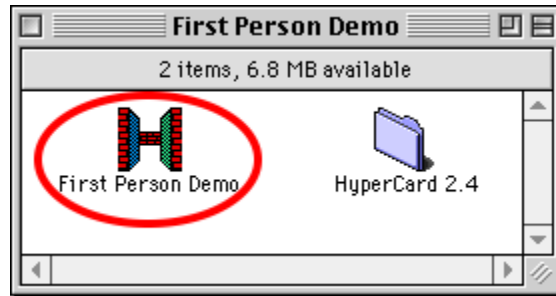
How Do I Play?

If using an emulator, follow the relevant guides to get started:

- <https://www.gryphel.com/c/minivmac/start.html>
- https://www.emaculation.com/doku.php/basilisk_ii_setup

Remember to attach the “First Person Demo” disk image, along side your boot disk.

Once you arrive at the Desktop, double click on the “First Person Demo” disk icon to view the contents, and double click on the “First Person Demo” file.



*When running on MinivMac, increase the emulator speed by holding Ctrl+S, and while still holding down Ctrl, press A for “All Out” speed.
Also try disabling “AutoSlow” in the same menu (W).*

Upon opening, the demo will detect the screen resolution of your machine, and set its internal settings for the best experience.

When the “Thinking...” message disappears after a few seconds, you should be at the Load Level Prompt.

There are currently 4 levels. Type in a number from 1 to 4 and click OK. The level should load after a few seconds.

The controls are:

- ↑ - Move Forward
- ↓ - Move Backward
- ← - Look Left
- → - Look Right
- z - Strafe Left
- x - Strafe Right

You can load a new level, or switch between modes that your machine supports, at anytime using the “Game” menu.

Game
✓ B&W Colour
✓ 512x342 512x384 640x480
Load Level

B&W - Draw all graphics in black and white (Colour machines can use this too!)

Colour - Draw all graphics in 8-bit colour, if your machine supports it.

512x342 - Set the stack resolution to 512x342.

512x384 - Set the stack resolution to 512x384.

640x480 - Set the stack resolution to 640x480.

Load Level - Load a new level to walk around in.

Increasing beyond the native resolution of your machine is not recommended, but decreasing the resolution is fine.

If you want to navigate through the cards in the stack, to see how things work, choose “Debug Mode” from the “Developer” menu. This will show all developer objects on the “game” card and “bg1” background, and will disable capturing key presses for player movement. The Left and Right Arrow keys will now move you through the stack.

Toggle Developer Mode again to return to gameplay.

Developer tools are used for working behind the scenes:

Developer
Message Box
Message Watcher
Variable Watcher
Navigation
captureKeyDown
✓ Text Arrows
Debug Mode
Bug Notes
Clear C&F
Blank B&W Btms
Recreate B&W Btms
Delete B&W Btms

Message Box - Toggles the Message Box

Message Watcher - Toggles the Message Watcher

Variable Watcher - Toggles the Variable Watcher

Navigation - Toggles the Navigation Palette

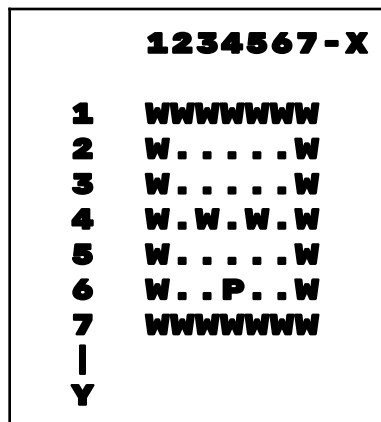
captureKeyDown - Toggles if key presses should be interpreted as player movement, or stack navigation.

TextArrows - Defines cursor and insertion point behaviour in HC.
Debug Mode - Toggles additional debug messages and visibility of objects.
Bug Notes - Quick access to the Bug Notes document.

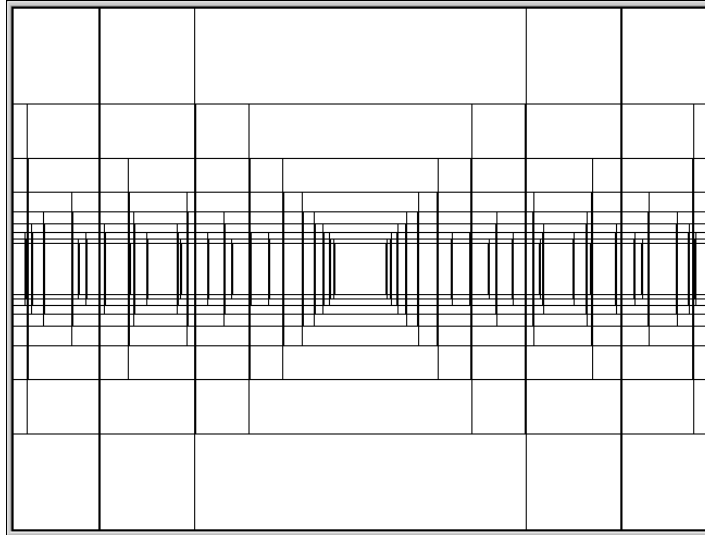
Clear C&F - Wipes the black and white ceiling and floor patterns from the view.
Blank B&W Btms - blanks out all textures displayed by the BWBtms.
Recreate B&W Btms - Destroys and recreates all BWBtms based on the currently loaded reference data.
Delete B&W Btms - Destroys all BWBtms.

How Does It Work?

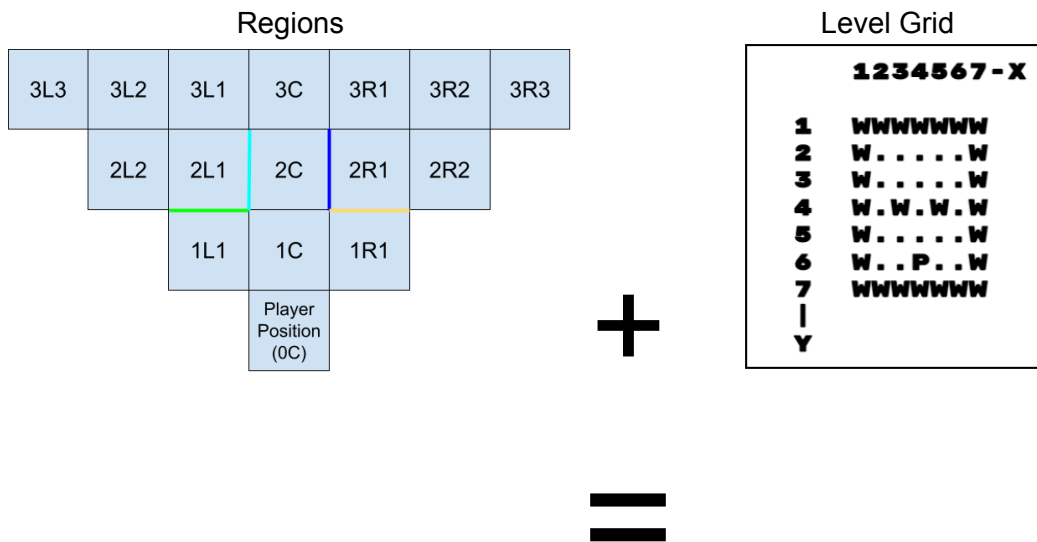
Levels are grid-based (like chess) to keep things simple. Cells in the grid can be one of several types (for now just solid opaque walls, walkable empty floor space, and the player). The player can step onto grid cells in front of, behind, and to the sides of themselves. They can turn to look in any of the 4 cardinal directions, so locked to 90 degree increments.

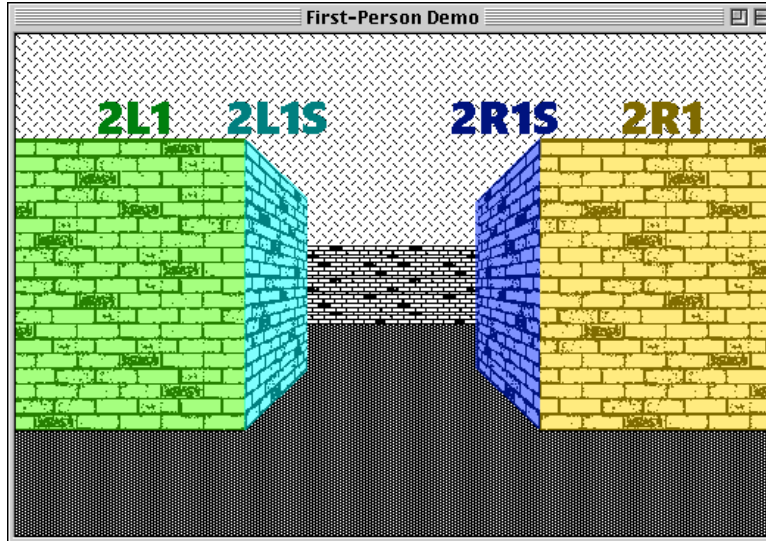


Because of these factors, the player's view of the world will always approximate something like this:



Different regions of the screen will be filled in depending on where walls exist on the grid, relative to the player's position. These regions on screen where potential walls may be drawn to, have been dubbed "Regions".





Whenever the player moves or turns, we calculate what walls they can see from their new viewpoint (other items and elements will be added in future revisions). We call this “calculating a scene”.

When calculating a scene, regions on the screen are matched with gridCoords in the level that they relate to. When a region matches with a gridCoord in the level marked as a wall, the texture information for that wall is queried.

The information regarding what elements the player can see, and where they should all be drawn on screen, is called the scene history.

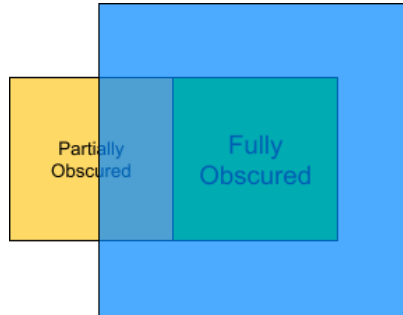
0L1S 3,6	2L1 3,4 w Brick t	4L1 3,2
0R1S 5,6	2L1S 3,4 w Brick t	4L1S 3,2
1C 4,5	2R1 5,4 w Brick t	4R1 5,2
1L1 3,5	2R1S 5,4 w Brick t	4R1S 5,2
1L1S 3,5	3C 4,3	5C 4,1 w Brick t
1R1 5,5	3L1S 3,3	5L1 3,1 w Brick t
1R1S 5,5	3R1S 5,3	5R1 5,1 w Brick t
2C 4,4	4C 4,2	

The scene history data that describes the scene, above.

The sky and ground are drawn separately from regions, as a background, filling in the rest of the screen space, and meeting at the horizon line in the centre of the players view.

The Visibility Problem

When the player is facing an opaque wall, it would be beneficial for the program to understand that the player cannot see any other objects obscured by that wall, and to not waste time processing them.



The primitive solution that has been employed here, is to compare every region with every other, and calculate those which are fully obscured by others. This job is performed offline, and resulting data is written to a look-up field referenced during gameplay.

```

1C 1L1S,1R1S,2C,2L1S,2R1S,3C,3L1S,3R1S...
1L1 2L1,2L2S,3L2,3L2S,3L3S,4L2,4L3...
1R1 2R1,2R2S,3R2,3R2S,3R3S,4R2,4R3...
2C 2L1S,2R1S,3C,3L1S,3R1S,4C,4L1S,4R1S...
2L1 2L2S,3L1,3L2,3L2S,3L3S,4L2...
2L2 3L2,3L3S,4L3,4L4S,5L4...

```

A small excerpt of the “exception” look-up field...

During the calculation of a scene, when a region has been determined to be an opaque wall, the look-up field is consulted for an entry pertaining to that particular region. Any regions fully obscured by that region are listed, and those regions are then *excepted* from being tested during the remainder of that scene calculation.

For this behaviour to work correctly, when calculating a scene, we begin translating the regions closest to the player first, then expand outwards towards the horizon. This ensures that any excepted regions (that are naturally further away from the player than the current region being translated) are black-listed from being translated, before the program naturally gives them the chance to be.

The scene is calculated in the following order:

- Immediately in front of the player first, then outwards, towards the horizon.
- For each step away from the player, the Center region is translated first (e.g. 1C), then the Left Hand Side of the view, then Right Hand Side.
- For each side of the view, the first “side” (trapezoid) region is translated first, e.g. 1L1S or 1R1S.
- Then the next “regular” (square) region, e.g. 1L1 or 1R1.
- Then repeat for side and regular regions outwards, e.g. 1L2S,1L2

The 0C region is skipped translating, because it is located behind the player’s head, and not visible in the scene.

Looking closely at the potential world view for the first person perspective, we can make several observations:

Regions on one side of the view will never obscure regions on the other side. I.e. 1L1 will never obscure any XRX regions.

More than one region on the screen can reference the same gridCoord in the level. The player could see the front face of a wall (visibly square, parallel to the player), and also the inner side of the wall (visibly trapezoid, and facing away from the player at 90 degrees).

Because both of these on-screen regions are referencing the same gridCoord, they must share the same state at all times. They are together both either an opaque wall, or empty floor space, but never one of each during the same scene.

Therefore, if one of those regions is translated to be an opaque wall, then we can assume that both regions will be. And if so, then any regions marked as *excepted* by one of those regions, should (by proxy) be labelled as excepted by the other region (because it is the one gridCoord marked as an opaque wall, after all.)

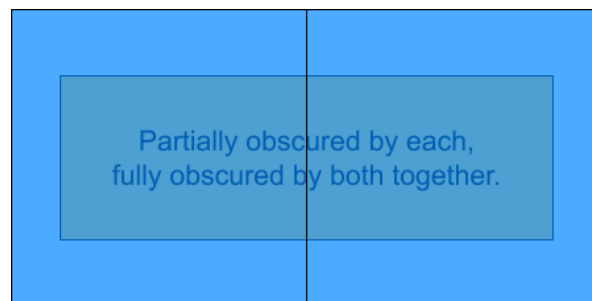
When a case like this is detected, the common excepted regions are processed once for both regions, reducing duplicated work.

The program contains checks for this case, and others, for increased efficiency.

A limitation of the current system is that excepted regions are referenced to other regions, not to the source gridCoord. This could be revised in a future release.

Unions

When a far away region is partially obscured by a closer region, it is still partially visible in the scene, and should therefore still be drawn. But what happens when a far away region is fully obscured by two adjacent near regions, but not fully obscured by any single one?



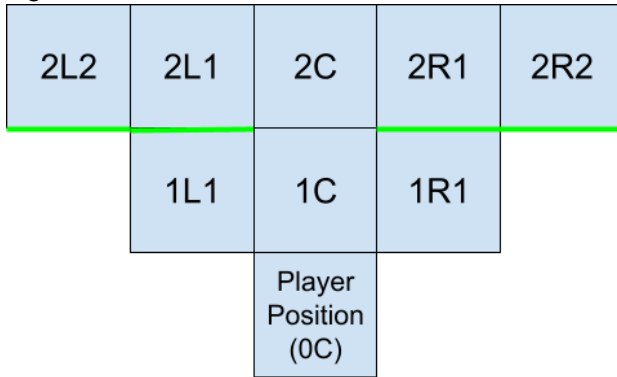
Of course it should not be visible in the scene, but it has not been marked as excepted by our current rule set.

We can refer to cases like these, when adjacent regions are both translated to be opaque walls, as region “Unions”. They are allowed to be treated as a single large region, and have its own case when calculating exceptions.

There are multiple types of Union:

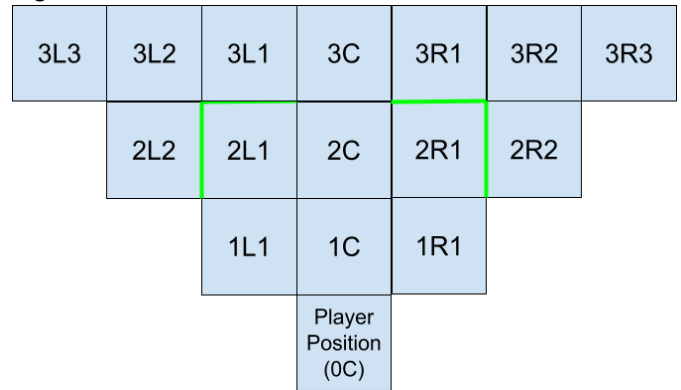
Adjacent Regulars

e.g. 2L2,2L1 or 2R1,2R2



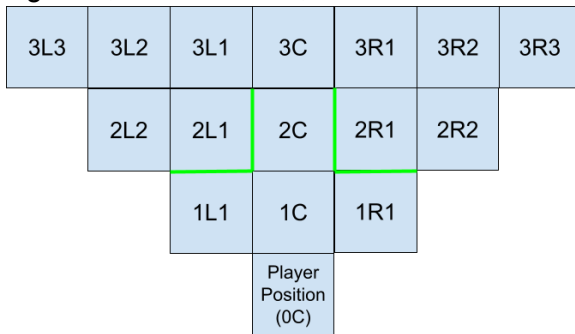
Convex Corners

e.g. 2L2S,3L1 or 3R1,2R2S



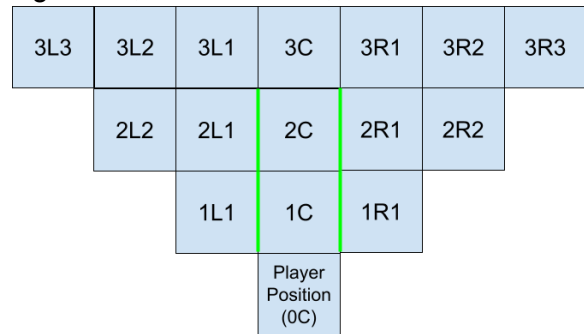
Concave Corner

e.g. 2L1,2L1S or 2R1S,2R1



Straight Line

e.g. 1L1S,2L1S or 2R2S,1R1S



Notice how the names of Unions differ between the Left and Right sides.

It was decided that the regions listed in unions should be ordered from Left to Right, from the players point of view.

$2L2 < 2L1 < 2R1 < 2R2$

$1L1S < 2L1S < 2R2S < 2R1S$

A naming convention had to be decided upon, for the code to act consistently in all cases.

This structure was chosen simply because, in HC, objects are drawn on screen from left to right.

As the scene is calculated, for each opaque wall detected, we check if this region could contribute to any of the 4 possible unions.

We use the scene history calculated thus far, to search for the existence of the other region that would complete the potential union.

Certain regions may only contribute to certain unions. e.g. Center regions are eligible to be part of Adjacent Regulars and Convex Corners, but not Concave Corners (because the 2nd region

would be a “Side”, and would be obscured by the center region) or Straight Lines.

It is also worth noting that when testing for unions, we can only compare with regions that have already been translated previously. e.g. for a potential Convex Corner, 3L1 must search for 2L2S in the scene history. When 2L2S is under test, it would be unable to search for 3L1, as it would not have been translated yet, and it’s state is uncertain.

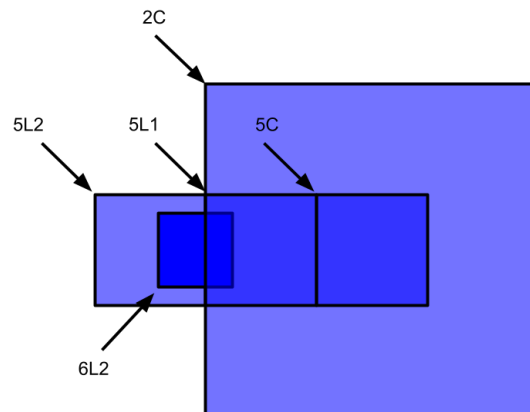
Early in development, it was observed that as regions shrink in size, as their distance from the player increases (towards the horizon), it was apparent that no far-away on-screen region was ever wide enough to span the width of two nearer adjacent regions.

This meant that unions of two adjacent regions were the limit of work required to catch these cases. It was not necessary to calculate unions of 3 or more adjacent regions.

Later on in development, it was observed that this may no longer hold true, as far away side regions are becoming increasingly wider. But instances of this problem occurring is rare, and often blocked due to other factors.

Special Cases

Consider the following case:



6L2 is fully obscured by the Adjacent Regular Union of 5L2,5L1, but is not fully obscured by either of them on their own. In fact it is not fully obscured by any single region in this example.

5L1 has been excepted by 2C, which is translated much earlier during scene calculation. 5L2 is still visible and free to be translated. It has been found to be an opaque wall, and it searches the scene history for 5L1, hoping to form a union with it, but 5L1 is not present. That union is not formed, 6L2 is not excepted, and is free to be translated, when it is not visible in the scene, wasting processing time. If it is found to be a wall, even more processing time is wasted drawing it on screen, as it will just be overlaid by nearer regions.

This happens because we have not declared a union type that could accommodate pairs of regions such as 5L2 and 2C. Implementing a case for every arbitrary combination of regions such as this would astronomically increase the cpu workload during scene calculation, during gameplay. Likewise, when an opaque wall is detected, looping through nested exceptions of

exceptions would also be impractical, as there would be many duplicate records.

Instead, we can adjust the rules of which regions in the scene history (or lack thereof) are eligible to be included in unions.

In the case above, we want to allow 5L1 to be included in a union, even though it is not visible in the scene, has been excepted from being processed and drawn to screen, and thus does not exist in the scene history.

We also want to allow regions that have been translated to be opaque walls to be included in unions.

At this point in development, considering all possible cases, we can see that the only case in which we DO NOT want to allow a region to be part of a union, is when it has been translated as transparent empty floor space.

We can change our rule set to allow any region, weather in the scene history or not, to be a valid contender for unions, as long as it has not been translated to be transparent empty floor space.

A revised method of exception processing is planned for a future release, that utilises a “tree” data structure for excepted regions, that is free from duplicate entries. This will allow for a smarter program flow that will hopefully cost fewer cpu cycles.

Due to the limitations of HC’s ability to draw graphics on screen, region textures must be drawn in their entirety. We are unable to draw partial regions or sub sections. To prevent corrupted views from occurring, HC allows for some amount of Layering.

While each scene is calculated from the player position, towards the horizon, the calculated scene is then drawn to the screen in reverse order, from the horizon towards the player position.

This ensures that when a region is drawn to screen that should only be partially visible, the closer / obscuring object is drawn afterwards, over the top of the previous object, partially obscuring it, and resulting in a correctly rendered scene.

Under the Hood

A basic understanding of HyperCard and it’s scripting language HyperTalk is required for the following sections.

Glossary

HC and Macintosh Specific Terms and Acronyms

bg - Background.

btn - Button.

cd - Card.

fld - Field / Text box.

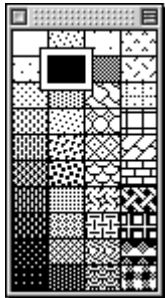
Handler - Similar to a conventional Function in programming parlance, Handlers are scripts that intercept *messages* as they are sent between objects in HC, and perform actions based on the specific event. E.g. when the stack opens, an “openStack” message is sent through HyperCard. Any openStack handlers present intercept this message and perform any commands you would like executed each time the stack opens.

Item - An item in a comma-delimited word e.g. “apple,banana,orange,grape” is a word, and “banana” is item 2 of that word.

Message - Are sent between different objects in HC (e.g. button, field, card, background, stack) as events occur (e.g. mouse click, key press, cardOpen, stackClose).

Objects - Anything that can hold a script, e.g. buttons, fields cards, backgrounds, stacks.

Patterns - Small black and white 8x8 pixel tiling textures, often featured in MacPaint and other Macintosh apps.



PICT - A vintage Macintosh picture file format. Supports both rasterised and vector images as well as text data. Originally supported only black and white images without transparency. Upon release of the Macintosh II series (the first in the Macintosh family to feature colour capable displays), the PICT file format was updated to support colour, and an alpha channel for transparency.

pt - Point, a single x,y coordinate.

Rect - Rectangle, in the format of leftSide,topSide,rightSide,bottomSide.
startX,startY,endX,endY is synonymous.

Resource - An intrinsic feature of the classic Mac OS architecture, Resources are additional supporting pieces of reference data bundled with native Macintosh applications and files. Resources are held in a file’s Resource Fork (in contrast to the Data Fork, where the binary code of the app / file resides).

Types of Resources include but are not limited to: images, icons, sounds, patterns, dialogs, fonts. The Resource Fork and the resources within can be edited with tools such as ResEdit, for customisation of apps and files.

Word - A string of letters and characters delimited by space characters. e.g. The 4th word of

this sentence is “of”.

Terms Specific to this Stack

Region - A rectangular selection of screen space where a *potential* wall is located. Regions have rects. Regions come in two types: *Regular* and *Side*.

Examples of regions:

- 1C
- 2L1
- 3R2S

Character 1 is the region’s distance from the player’s position (in steps).

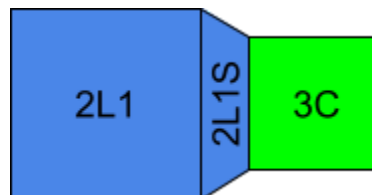
Character 2 indicates is the side where the region is located, from the player’s perspective, (L)eft, (C)enter or (R)ight.

Character 3 indicates the “shoulder” or number of steps the region is located to either side of the player. This character is unused for Center Regions.

Character 4 indicates if a region is a “Side”. Regular regions are the default case.

Regular - A perfectly square region / rect portraying the *front* of a wall, from the players point of view.

Side - A trapezoidal region portraying the *side* of a Wall, from the players point of view. Side regions link the regular regions of one Distance, to those in the next following distance.



Side regions have a nearHeight and a farHeight. The nearHeight is the same as the regular regions in the same distance (in fact, the *Unit* of the distance). The farHeight is the Unit of the next distance, further away from the player. The width of side regions are determined by the screen coordinates of their connecting regular rects.

Rect - The rectangular pixel coordinates of a region, on screen.

“Inside” Regions - Regions that are at least partially within the viewRect, and are therefore visible from the player’s point of view.

“Outside” Regions - Regions that exist fully outside the viewRect, and are therefore not visible from the player’s point of view.

“Near” Regions - Regions that are a relatively close distance to the players position e.g. 1C, 2L1.

“Far” Regions - regions that are a relatively far distance from the players position e.g. 6R2, 7L1.

BWBtns - A family of button objects belonging to background “bg1” that enable the use of black and white graphics. A single BWBtn exists for each region to be drawn on screen. These buttons are created based on the data present in the regions_etc field, their size on screen is set to the region’s rect. The button’s style is set to transparent, and their icon ID set to -1. This allows the btn to adopt the image of a PICT image resource held in the stack’s Resource Fork, where the given name of the BWBtn Object, matches a PICT Resource.

The list of currently existing BWBtns is kept in field “regions_btnIDs” on card “game”. BWBtns are resized when the resMode changes, and new regions with varying rects are imported.

C&F - Ceiling and Floor.

Curline - “current line”. A pointer used when looping through multiple lines in a field.

“Compressed” Points - The two corners of a trapezoidal side region that are “compressed” inward from their usual rectangular points.

Distance - The number of “steps” or gridCoords directly in front of the player. Distances have Units.

Exception - A region that is fully obscured by another region, from the players point of view. Excepted regions are “excepted” from being drawn to screen, as part of a calculated scene.

Grid - The currently loaded level as seen from a top down perspective. The grid is a paragraph of text, in a field (best viewed with a monospaced font).

GridCoord - The x,y coordinate of a text character in the currently loaded grid. The “xth” character (from the left) of the “yth” line (from the top) in the grid text field.

Perspective Curve - A mathematical formula defining the relationship between the size of the regular regions on screen (measured in pixels) in each distance, as they are positioned increasingly further away from the player, towards the horizon.

See the “[View Perspectives](#)” chapter for more info.

RUT - Region Under Test. The current region being tested / translated during the calculation of a scene.

Scene - A single pass render of the first person view of the world, in front of the player. A new scene is calculated for every step or turn the player moves.

Shoulder - The number of steps away a region is, to the left or right of the player’s position. There exists a set number of shoulder regions in each distance.

Texture - A picture overlaid onto a wall.

Union - A “super region” consisting of 2 adjacent regions. There are multiple types of unions,

each one based on the relative positions of the joined regions. See the “[Unions](#)” chapter for more info.

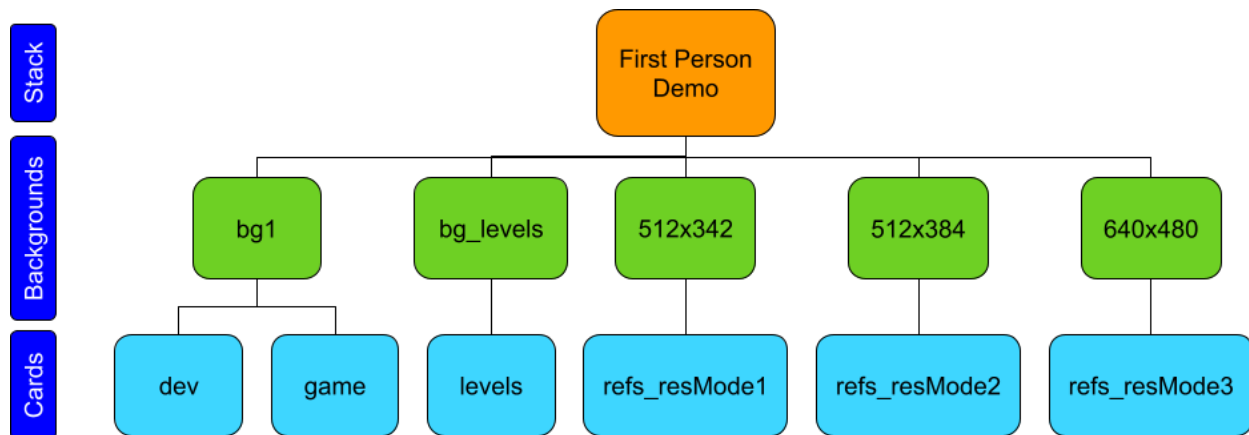
Unit - The length in pixels (width / height) of the square regular regions in each distance.

viewRect - A rectangular pixel region on screen wherein the first person perspective is to be drawn. This is often matched to the full screen resolution of specific Mac Models. ViewRects begin from pixel 0. E.g, the viewRect of a 640x480 monitor is 0,0,639,479.

Stack Structure

Only the more significant handlers and variables are discussed below. There are many more small handlers and variables throughout the program that are self explanatory.

The Stack is currently about 2MB in size; 1.8MB is taken up by resources (including the three colour textures sets, and the one B&W texture set), while the data fork, including all scripts, is 200KB.



Global Variables

captureKeyDown - Bool - Whether to capture key presses to enable the player to move around the level (1), or disable player movement input, in order to navigate through the stack cards (0).

cCol - RGB - The RGB value to be used when drawing the ceiling in colour mode.

colourMode - Bool - Whether to draw in black & white (0), or 8-bit colour (1).

cPat - Int - The number indicating the B&W Pattern to be used when drawing the ceiling in B&W mode. (1-40)

debug - Bool - Debug Mode enabled or not. Causes additional status messages when computing, and shows / hides developer objects.

fCol - RGB - The RGB value to be used when drawing the floor in colour mode.

fPat - Int - The number indicating the B&W Pattern to be used when drawing the floor in B&W mode.(1 - 40)

maxDistance - Int - Controls draw distance. The distance in front of the player that should be computed and viewed from the first person perspective. Usually set to 6 or 9.

numOfLevels - Int -The number of levels currently kept on the “levels” card. When new levels are added, update this number. Used during level loading process.

playerCoord - Pt - The player’s x,y coordinate in the level / grid. The character at this gridCoord is edited to “P”.

playerFacing - Char - “n,e,s,w”. The cardinal direction the player is facing. It is currently hard coded that the player is facing north (towards the top of the grid) when a level is first loaded.

resMode - Int - Current Resolution Mode.

1 = 512x342,
2 = 512x384,
3 = 640x480.

viewRect - Rect - The specified pixel rectangle on screen where the first person view should be drawn. Relative to the Top Left corner of the stack window. E.g. 0,0,639,479

Stack Script

Handlers:

openStack

Sets up several important variables and states:

- debug
- maxDistance
- visibility of developer objects
- detects screen resolution in order to auto-setup viewRect and colourMode
- creates custom menus
- calls the Init handler to continue setting up viewRect.
- evaluates existing BWBtNs, and if they need to be resized due to new resMode.

closeStack

Cleans up stack before closing, in preparation of it opening again, potentially on a different machine model with different capabilities.

Blanks out any images presented by the BWBtNs, scrubs the C&F designs for both colour and B&W systems, for a blank presentation on next launch.

developerMenuHandler

+ un-handler, + items, + messages.

Creates custom Developer Menu

gameMenuHandler

+ un-handler, + items, + messages.
Creates custom Game Menu

aboutMenu

Creates custom stack "About" window.

colourModeOff

colourModeOn

Performs tasks / cleans up when moving between Colour & B&W Modes.

set512x342

set512x384

set640x480

Performs tasks / cleans up when moving between resModes.

promptLoadLevel

Defines User prompt and input sanitising when asking which level to load.

Backgrounds

bg1

Handlers:

init

- Gets players position from the grid,
- sets player facing direction and supporting structures,
- sets window size and viewRect based on resMode,
- restores reference data (regions, rects, exceptions etc) based on resMode.

drawBackground

scrubBackground

Draw and scrub colour background C&F.

checkGridCoord

Occasionally when inspecting coordinates of characters in the grid field, the x (character) and y (line) pointers may wrap around to the other sides of the grid, or worse extend into non existent negative coordinates (beyond the top left corner of the grid). This handler filters these cases out, and prevents garbage data from passing through to the rest of the program.

restoreRefs

Restores the relevant reference data (regions, rects, exceptions, etc) when changing resModes.

Fields:

regions_etc

Lines are in the format:

- word 1 - [regions](#)

- word 2 - **rects**
- word 3 - **sizes**
- words 4 & 5 - **Top and Bottom compressed-screen-pts** (if side region)
- words 6 & 7 - **Top and Bottom compressed-local-pts** (if side region)

E.g.

- **0L1S** 0,-80,79,559 80,640 79,-1 79,480 79,79 79,560
- **3C** 218,138,421,341 204,204

Lines for regular regions are limited to the region, rect, and size.

Lines for side regions have those, and also compressed screen-pts and local-pts.

Compressed pts are the two corners of a trapezoidal side region that are “compressed” inward from their usual rectangular points.

These come in two types:

- Screen-pts are where the compressed pt is located on the screen, relative to the viewRect.
- Local-pts are where the compressed pt is located in the texture image file, relative to the top left corner of the image. Compressed local pts are referenced during texture creation.

The first of the two pts of that type, relates to the top corner.

The second of the two pts of that type, relates to the bottom corner.

rects_old

An old back up of regions_etc containing regions that exist fully outside of the players viewRect. These extra regions are needed during the calculation of Excepted regions.

sorted_draw_backup

A copy of regions_etc whose lines have been sorted for optimised scene calculation.

Regions are ordered from near to the player, to far away.

The advantage of this sorted list is that for every region, any other regions that it fully obscures, are listed beneath said region in the list. This speeds up time spent during processing.

distances_maxShoulders

Noting the maximum shoulder generated for each distance.

ref_single_excepts

ref_union_excepts

Lists containing all regions, and all other regions that they fully obscure (except).

The single list is for single regions i.e.the rect of 3C excepts 3L1S,3R1S,4C,4L1S,4R1S, etc.

The union list is for unions of 2 regions, and any other regions they except together e.g. the combined rects of 3L1,3C except 3L1S,3L2S,3R1S,4C etc.

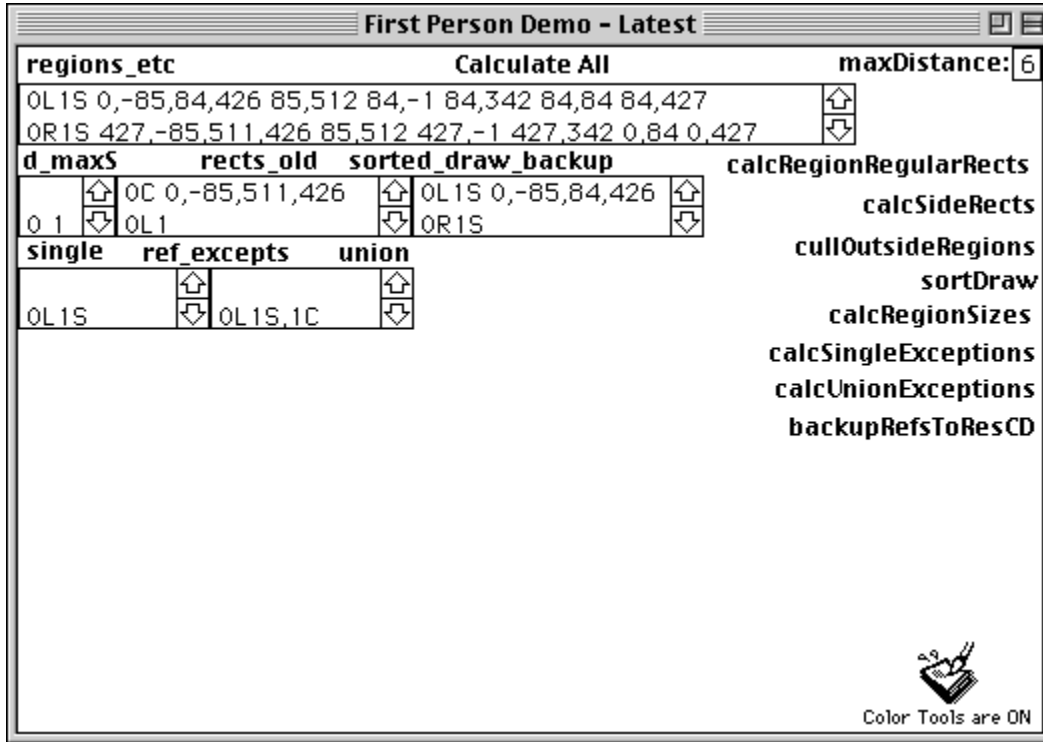
bg_levels

Contains level data in the form of grid layouts and wall / texture information.

You can edit these fields and add additional levels, by incrementing the numbers of the field names. See the “[How to Add & Edit Levels](#)” chapter for more info.

Cards

dev



cd Handlers:

does1cover2

Compares rects of regions to detect partial or complete overlap. Used to calculate excepted regions.

Buttons with Handlers:

calcRegionRegularRects

Calculates the Unit, regular regions / shoulders for each Distance, based on the viewRect, maxDistance and the perspective equation. writes to the regions_etc field.

We continue creating regular shoulder regions outwards, until either:

- one is calculated to exist fully outside of the viewRect, or
- we have reached 9 shoulders for this side, and are unable to address any more before our single character integer rolls over.

When this occurs, we proceed to the next distance.

calcSideRegions

Calculates the rects for all side regions based on existing regular regions. writes to the regions_etc field.

cullOutsideRegions

Backups the current contents of the regions_etc field which contains regions that exist fully outside of the viewRect. Then culls these “outside” regions from the regions_etc field.

sortDraw

Copies the (post-culled) contents of regions_etc into the sorted_draw_backup field, and sorts the lines by the following order:

sort lines of bg fld “sorted_draw_backup” by (char 1 of word 1 of each) & (char 2 of word 1 of each)

calcRegionSizes

calculates the size (width,height), screen-pts, and compressed-pts of regions in regions_etc. For side regions, the size is of the outer rectangle (not including compressed trapezoid points). The size and pts are calculated to assist during texture creation. See regions_etc for more info.

calcSingleExceptions

Compares each single region rect to every other single region rect, in order to calculate those that are fully obscured (which then become excepted from drawing to the screen, during a scene). Writes to the ref_single_excepts field.

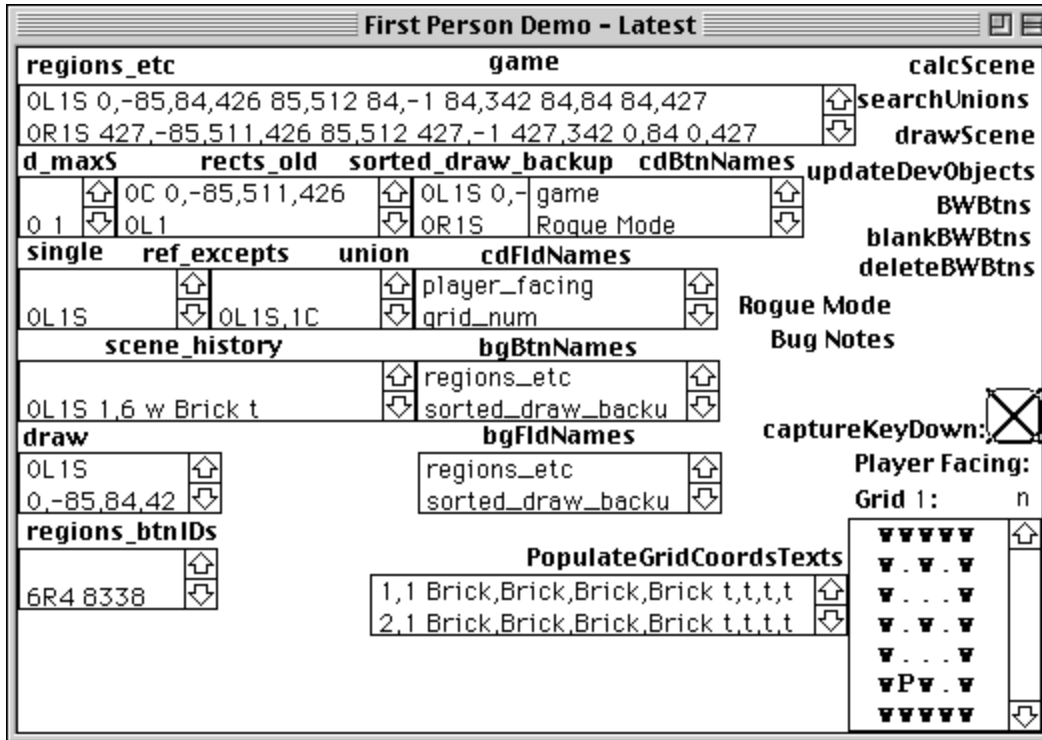
calcUnionExceptions

Checks for valid unions of regions in the regions_etc field, and compares the combined rect of those regions to every other region rect, in order to calculate those that are fully obscured (which then become excepted from drawing to the screen, during a scene). Writes to the ref_union_excepts field.

backupRefsToResCD

After the tasks above have been run, this handler will backup the newly generated reference data to the relevant resMode card. This reference data will then be restored to the respective fields on bg1, when the resMode is changed, The gameplay functions utilise this reference data.

game



cd Handlers:

keyDown

Specifies key codes that map to player movement. Works with captureKeyDown flag in order to capture these key input events.

clockwise

counter

Called when player turns 90 degrees right or left respectively.

movePlayer

Called when player takes a step forward, backwards or to either side.

Gets the Type of gridCoord the player is attempting to move onto, and checks for collision.

If player attempts to move into a wall, a "That's a wall!" message appears, and no further action is taken.

translate

getGridCoord

calcCoord

calcCoordSkipX

calcCoordSkipY

getTextureOpacity

These handlers provide the connection between the regions that make up the player's first person view, with the level data. Translates an on-screen region (e.g. 3L1) to match with the corresponding gridCoord in the level data (e.g. 5,3).

calcSide

Determines which side of a wall (n,e,s,w) in the scene is visible and facing the player.

writeSceneHistory

Writes the data describing the scene being calculated in front of the player, to the scene_history field.

checkValidUnion

searchUnions

addSceneExceptionsBasedOn (region)

Ensures that potential unions exist in the scene_history field, before any exceptions are added based on this union.

removeFromDraw

Removes excepted regions from the draw field, thus preventing them from being tested / translated as part of the scene being calculated.

Buttons with Handlers:

calcScene

This handler copies the contents of the reference field sorted_draw_backup into the draw field. As the regions closest to the player are placed at the top of the list, they are tested / translated first. If they are found to be opaque walls, then any other regions that are excepted based on this region under test (RUT), (which in this case is likely to be most other regions), are removed from the draw list, before they get the chance to be tested / translated. The draw list shrinks as excepted regions are removed from the list, thus prevented from ever getting the chance of being drawn to the screen. The time taken to fully process the entire draw list diminishes as excepted regions are removed.

drawScene

Once a new scene has been fully calculated and written into the scene_history field (ordered from nearRegions to farRegions), this handler acts upon the items recorded in the scene_history field IN REVERSE ORDER (renders scene from far away to near the player position). This assists with the proper layering of regions as they are drawn behind / in front of each other. drawScene contains functionality for drawing the scene in both colour and B&W graphics, based on the colourMode.

updateDevObjects

This handler shows / hides all developer objects (buttons, fields etc) that belong "under the hood".

BWBtns

This button contains many scripts for managing the BWBtn Family.

Scripts are included to evaluate, resize, destroy & recreate each button dedicated for presenting the black and white graphics.

Bug Notes

A simple document for keeping track of developer debugging notes.

Fields:

grid

Contains the top-down view of the level layout

scene_history

Contains the information defining what walls in the level are and aren't visible, in the latest scene that the player can see from their perspective.

draw

A scratch field used during scene calculation.

regions_btnIDs

Keeps track of the IDs of all buttons in the BWBtn family, used for presenting black and white graphics. It is important that the IDs of these button are not confused with those of any other buttons on the background, as the BWBtNs are frequently edited and deleted as colourMode and resMode changes.

Other Misc Backgrounds and Cards

Reference Data Backgrounds - Cards:

- 512x342 - refs_resMode1
- 512x584 - refs_resMode2
- 640x480 - refs_resMode3

These backgrounds and cards contain backed-up reference data including regions, rects, and exceptions for each resMode. Relevant data is restored to bg1 when resMode changes.

Calculating Reference Data

You should only need to recalculate the reference data, if employing a new perspective curve, or if maxDistance has changed.

If you need to do this, enable Debug Mode, navigate to the “dev” card (the last card of the stack, one card to the left of card “game”), and click on the “Calculate All” button at the top of the card.

It is best to recalculate data on an emulated system, as the work may take significant time when run on authentic hardware. User prompts will appear during the calculation process, warning of this.

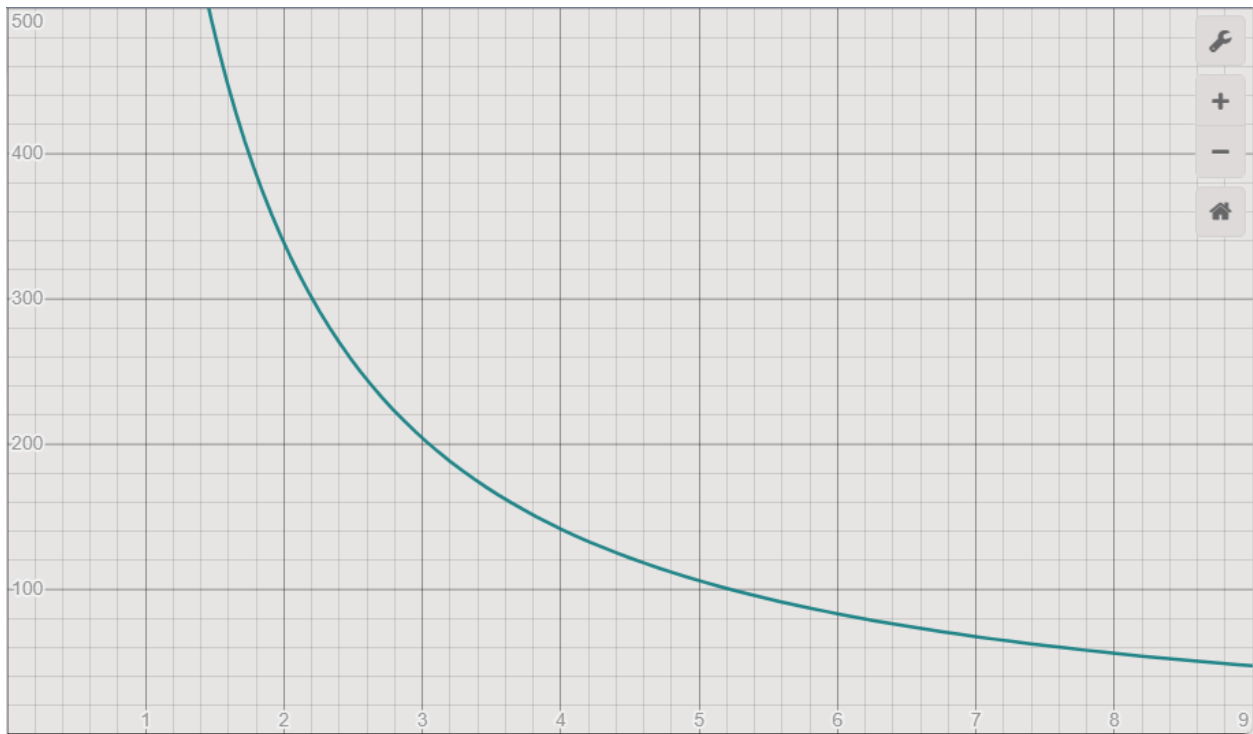
Run the Calculator once with each resMode activated, to ensure all data gets generated.

View Perspectives

A mathematical formula defines the relationship between the size of the regular regions on screen (measured in pixels) in each distance, as they are positioned increasingly further away from the player, towards the horizon.

The formula used for a 640x480 viewRect is:

$$y = \frac{400x^{-1.2}}{0.5} - 10$$



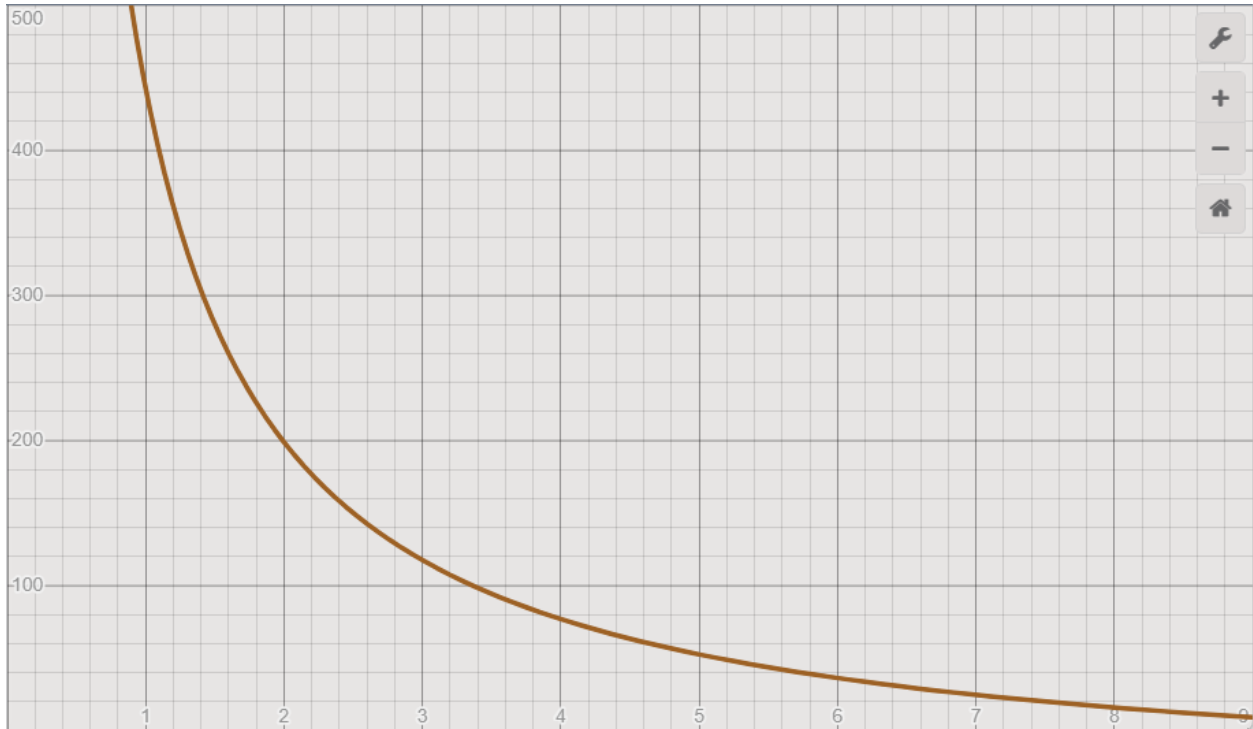
Where y = height = width of square regions, and x is the distance away from the player towards the horizon, in steps.

This curve greatly affects the player's perspective of the scene. It is not directly tied to the "viewing angle", but adjustments result in a "Dolly Zoom" cinematic effect. Fine tuning of the equation is necessary to achieve a realistic, and comfortable outlook of the scene.

An unsuitable curve would change the apparent visual spacing between equal-spaced objects as they are placed in a line, in front of the player, receding into the distance. Examples of this are discussed [below](#).

A different equation has been tuned for the 512x342 resolution of Classic Macintosh screens:

$$y = \frac{195x^{-1} - 30}{0.4} + 30$$



The reason for having a different equation, is due to the aspect ratio of the viewRect.

ResMode 1 and 2 share the same horizontal screen resolution, only the vertical resolution is marginally increased. For this reason, they have differing D0 and D1 values, but follow the 512x342 perspective curve values from D2 onwards. See "[About Shared Textures](#)" for more info.

These equations are applied only from distances 2 to 9.

In distance 0 (which relates to the side regions visible in the players peripheral vision), the height and width of square regular regions (the theoretical regions 0L1,0C, and 0R1) are forced to the *width* (in pixels) of the first person view window on screen, e.g. 640.

In distance 1 (which relates to regions immediately in front of the player), the height and width of regions are forced to the *height* (in pixels) of the first person view window on screen, e.g. 480.

This assumes a wide view ratio, as the regions in D1 could exceed those of D0, if the viewRect becomes portrait orientated.

Distance 1 is fixed so that when a player is standing in front of a wall, the entire wall texture is visible to them, without being cut off screen. This is to assist when interfacing with Terminals and other interactable objects (to be implemented in a future release).

The values of the curve at each distance are then rounded up or down so that they end on the nearest multiple of 4 (0,4,8,12 or 16). This is needed because when these numbers are divided by 2, we want the resulting numbers to be even (ending in 0,2,4,6,8).

When values are divided by 2 and end with 1,3,5,7, or 9, their regions appear asymmetrical and uneven when drawn on screen.

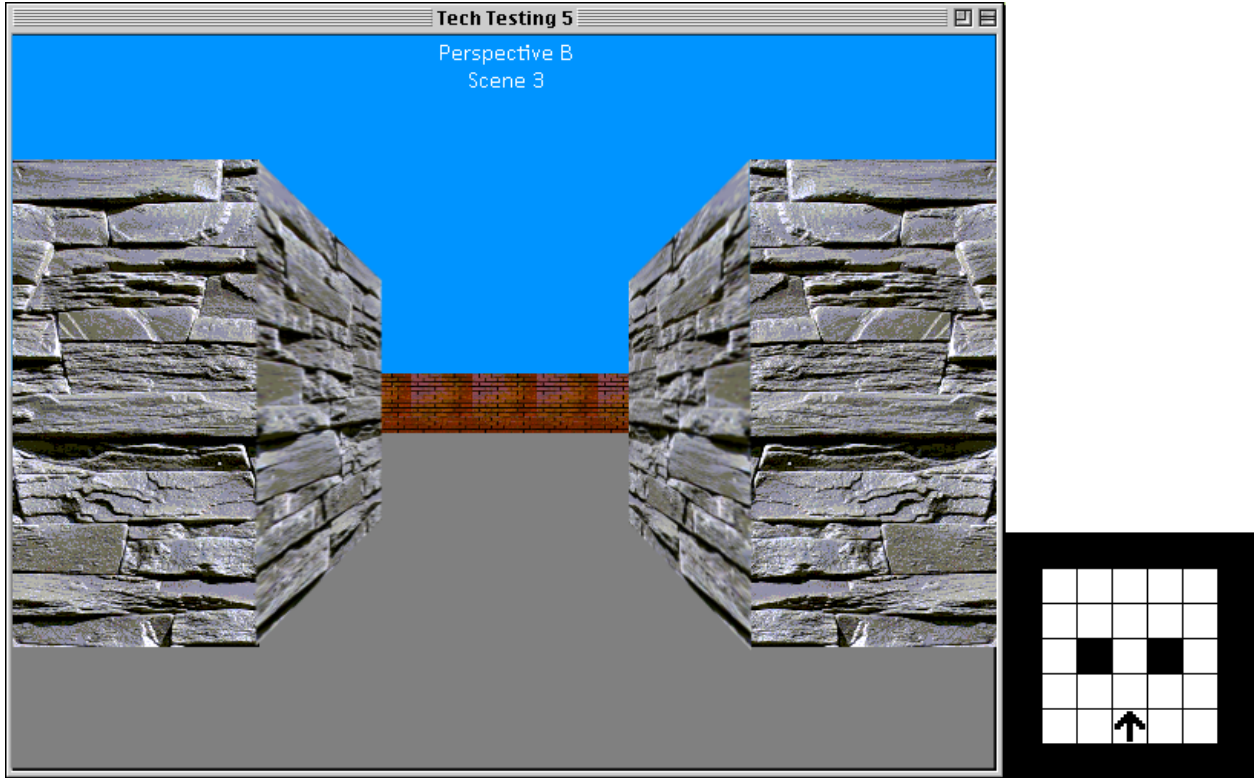
Ideal Characteristics of a Perspective Equation

- 9 is the greatest distance we can address with a single character. This should be the last distance that the player could see, as distance 10 should be at the horizon, with a region size of 0 pixels.
- Y should decrease gently, but exponentially, as distance increases. A linear decrease appears unnatural.
- It does not make a big visual impact if the fixed values of D0 and D1 do not follow the steep incline of the curve.

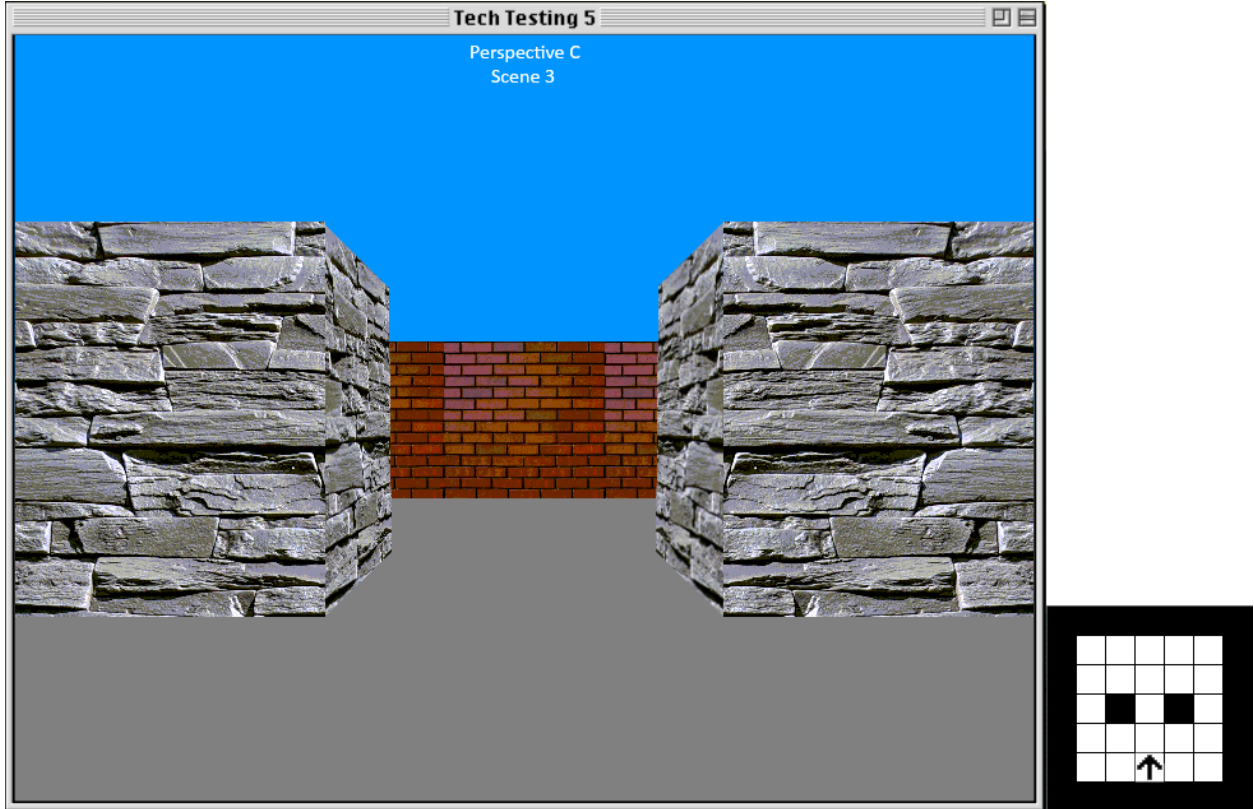
Here are some less than ideal perspectives generated during testing:



D2 is perceived to be far away from D1, with space between distances diminishing as distance increases. There is 1 step between the player and the pillars, and 2 steps between the pillars and the back wall at D5.

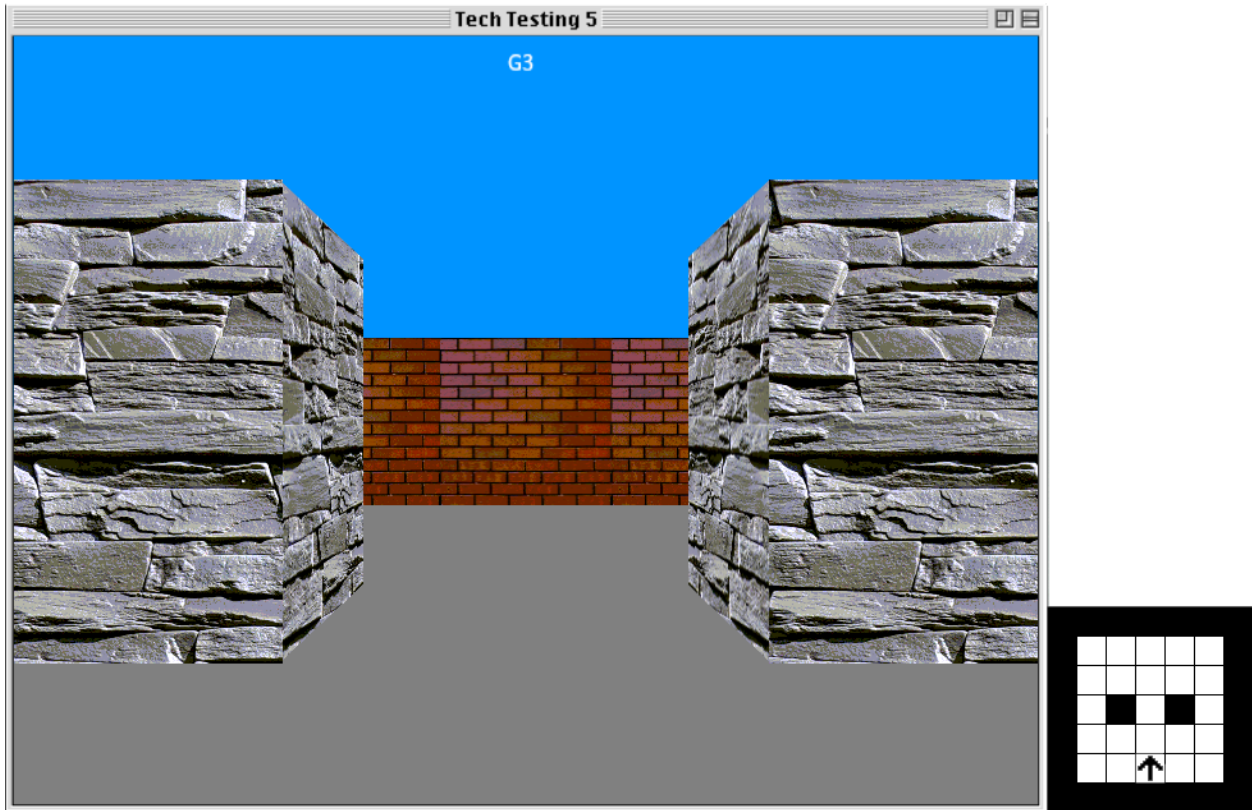


The opposite effect. D2 appears a comfortable length to D1, but then space appears to increase greatly as distance increases.



A big improvement. D2 still appears slightly too far away from the player, and D2 to D5 doesn't

seem far enough.



The chosen curve for 640x480.

About Shared Textures

The naming convention for texture resources is:

{BW or C} | {resMode} | {Texture Name} | {Side Info}

E.g:

- C1Brick
- C1Brick1LS
- C1Brick3RS

- BW1Brick0
- BW1Brick1R1S
- BW1Brick4
- BW1Brick4R3S

For Colour Textures:

- Only a single square texture is needed for all regular regions through all distances, as it can be scaled. It is recommended to set the height and width of this texture to the resolution of 0C, or the width of the viewRect, in both dimensions. This would ensure the

texture offers the maximum quality that the resMode is able to produce.

- No side-info characters are required for the single square texture, i.e. "C1Brick" is enough.
- The distance, side (L or R) and "S" characters are needed for textures of side regions, and are appended after the texture name. The shoulder character is not required because a single side texture can be stretched horizontally for each side region per side (L or R), without affecting the vertical placement of the compressed trapezoid points.
- It is recommended that side textures be the resolution of the maximum shoulder side in its distance (the widest one). This ensures the texture is the maximum size and quality that can be displayed for each side region in that distance. It will be scaled down when drawn to thinner regions.
- A 2nd horizontally mirrored resource is required for the other side (L or R) as colour PICTs are unable to be drawn in reverse, or mirrored. PICTs can only be drawn from top left to bottom right.

For Black & White Textures:

- 1 square texture resource is needed, per distance. It can be shared among all regular regions in that distance, but each distance will need a unique one.
- All side textures require unique resources, because the process that draws B&W graphics to the screen is unable to scale textures at all. Due to this, the distance, side (L or R), shoulder and "S" characters are required to follow the texture resource name.

All side region texture resources are unique to a perspective, as this affects the location of the compressed trapezoid points. Side textures do not align correctly when used in a resMode or perspective they were not designed for.

Because resMode 1 and 2 share the same perspective curve values, most (but not all) of their region sizes are identical. The draw handler takes this into account, and masks the names of those shared texture resources, so that duplicate resources with varying names can be avoided.

ResMode 1 and 2 share the same perspective curve from Distance 2 onwards.

The Unit for Distance 1 is fixed to viewHeight, which changes between those resModes, so the dimensions of rects in that distance differ.

Side Regions in Distance 0 have their "compressed" points fixed to the unit of Distance 1, so they also differ.

Based on the age of Macintosh models that this Demo is designed for, colour textures should be limited to an 8-bit colour depth.

An 8-bit image has:

3 bits assigned to the Red channel (8 shades),
3 bits assigned to the Green channel (8 shades),
and 2 bits assigned to the Blue channel, (4 shades).

It is recommended to “Posterize” your textures to ensure this colour depth.

Many posterize tools talk about “levels”. The number of levels per colour channel are the number of shades of that colour.

So to ensure our textures conform to 8-bit colour, set the Red and Green channel to 8 levels, and the Blue channel to 4 levels.

8-bit colour graphics work best when running on Mac OS 7 or later. It was found that when running a colour System 6 environment, not all colour texture resources were drawn to screen. However, the same textures do draw to screen when running under Mac OS 7 on the same emulated machine, with the same amount of RAM (Macintosh II with 8MB RAM). System 6 appears to be happier with 16-colour, or black & white graphics.

How to Add Additional Textures

If you wish to add additional textures to the game, the following tools are recommended:

Modern PC Tools:

- Paint.NET with the “Distort This!” plugin

Vintage Mac Tools:

- Adobe Photoshop version 2, 3, or 4.
- Quicktime 3.0 + Picture Viewer + BMP Plugin
- ResEdit (or Super ResEdit 2.1.3)

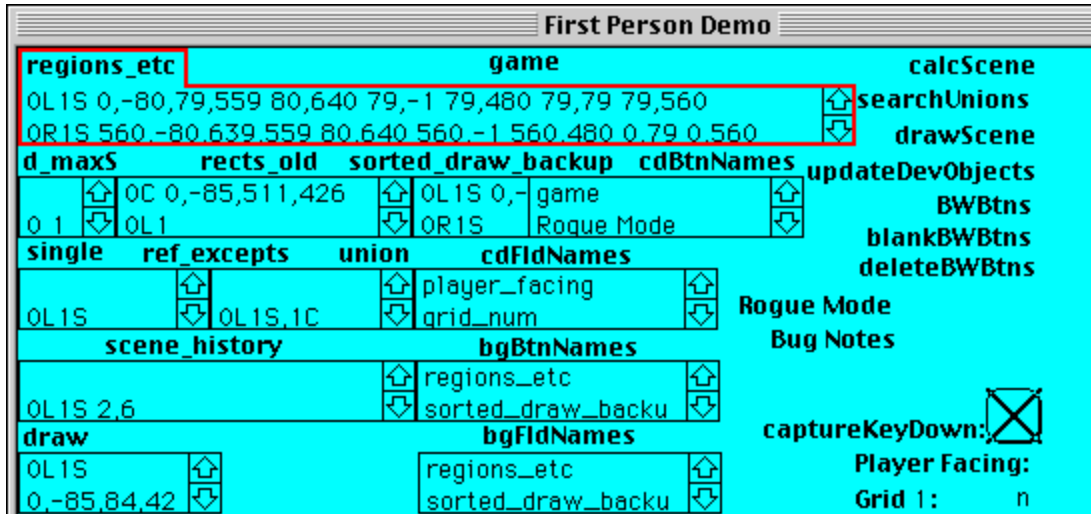
Firstly, decide for which resMode you want to add textures for, and if you want to create 8-bit colour, or black and white textures.

We require the region data for that resMode. Open the Stack, and click “Cancel” on the Load Level prompt.

Change to the relevant resMode with the commands in the “Game” menu.

Enable “Debug Mode” from the “Developer” menu, This will also disable key capture for player movement, letting you interact with the stack elements.

The card should fill with many buttons and fields. The top left field should be labelled “regions_etc”.



The lines in this field contain the pixel size of each region. This information is required when making suitable textures for regions. Refer to the “regions_etc” section under the [Backgrounds](#) section for the format of this data.

These fields exist on the Background layer, and are set to share common text between all cards of that background. In order to select the text to copy it, choose “Background” from the “Edit” menu, or press Command + B. The menu bar will change design to indicate you are in background editing mode. The text in the fields will now become selectable. Click the mouse inside the field to drop the insertion point. Press Command + A to select all text, and then Command + C to copy to clipboard, or use the menu options.

If using an emulator, check which key on your keyboard the Command Key is mapped to. It may be Ctrl, Alt, or Windows. Basilisk II users should be able to paste the text into an editor on your host system, as that emulator supports a shared clipboard.

If you are running the stack on a vintage machine or MinivMac, paste the text into a text editor such as Simple Text. It may be best to maximise the text editor window and take several screenshots with your phone, so that you have this information on hand.

Back in HC, to leave the background layer, Choose the edit background option again to return to the card.

Now that we have our reference data from the “regions_etc” field, we can begin creating textures on our host machine.

Square “Regular” Textures

It is best to create all square textures first, as they will be used during the creation of side textures:

- Create or import a base image, do any tidying up necessary.
- Then reduce it down to the square dimensions of the region in question.

- Posterize the image down to the correct colour depth.
- If making a black and white texture, remove colour information first, and use the brightness and contrast adjustments to find a pleasant threshold between the black and white levels. Setting the Contrast to 100% and the Brightness between 60-70% produced good results. Alternatively, use a Dithering plugin.
- Edit it so that the textures tiles nicely when placed side by side,

It is recommended to make the top and bottom of both regular and side textures, a solid 1-pixel-width black line. This helps provide contrast between the wall textures and the ceiling and floor. Leaving the left and right sides of the texture will allow them to tile nicely.

- Export as a BMP file. Paint.net will give you another chance to specify the bit depth there.

Trapezoid “Side” Textures

When creating Side textures in black and white, there must be a unique texture resource for every side region listed in the “regions_etc” field.

But when creating Side textures in colour, a single texture resource can be shared among all side regions in one distance, on one side of the screen (L or R).

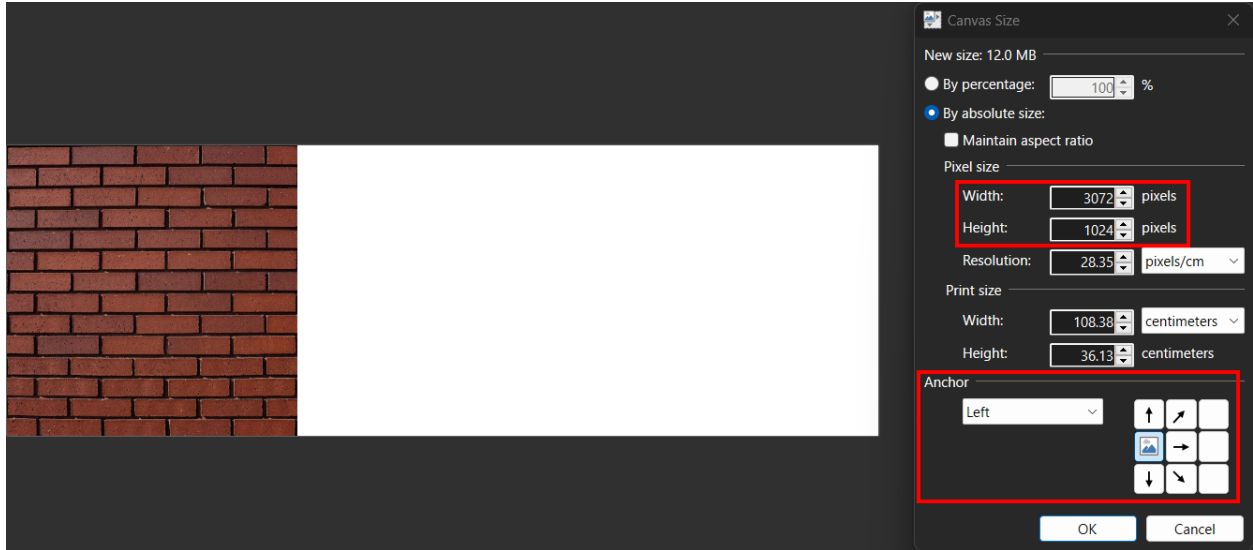
In this case, it is recommended to create the texture based on the furthest-out (widest) side region in that distance. This ensures the texture is the maximum size and quality that can be displayed for all other side regions, and will be scaled down width-wise when drawn to thinner regions.

Begin by opening the square texture for the same distance, or scale a larger square region down to that size.

When creating black and white textures, utilise the colour texture files for editing, as 1-bit images become corrupted when scaling.

Side textures of far-away distances (usually distance 3 or greater) will often be a wider horizontal resolution than the square texture for regular regions in that distance. When this occurs, adjust the width of the canvas size to 2 or 3 times the width of the square texture, and “Anchor” the image to the left or right side of the canvas.

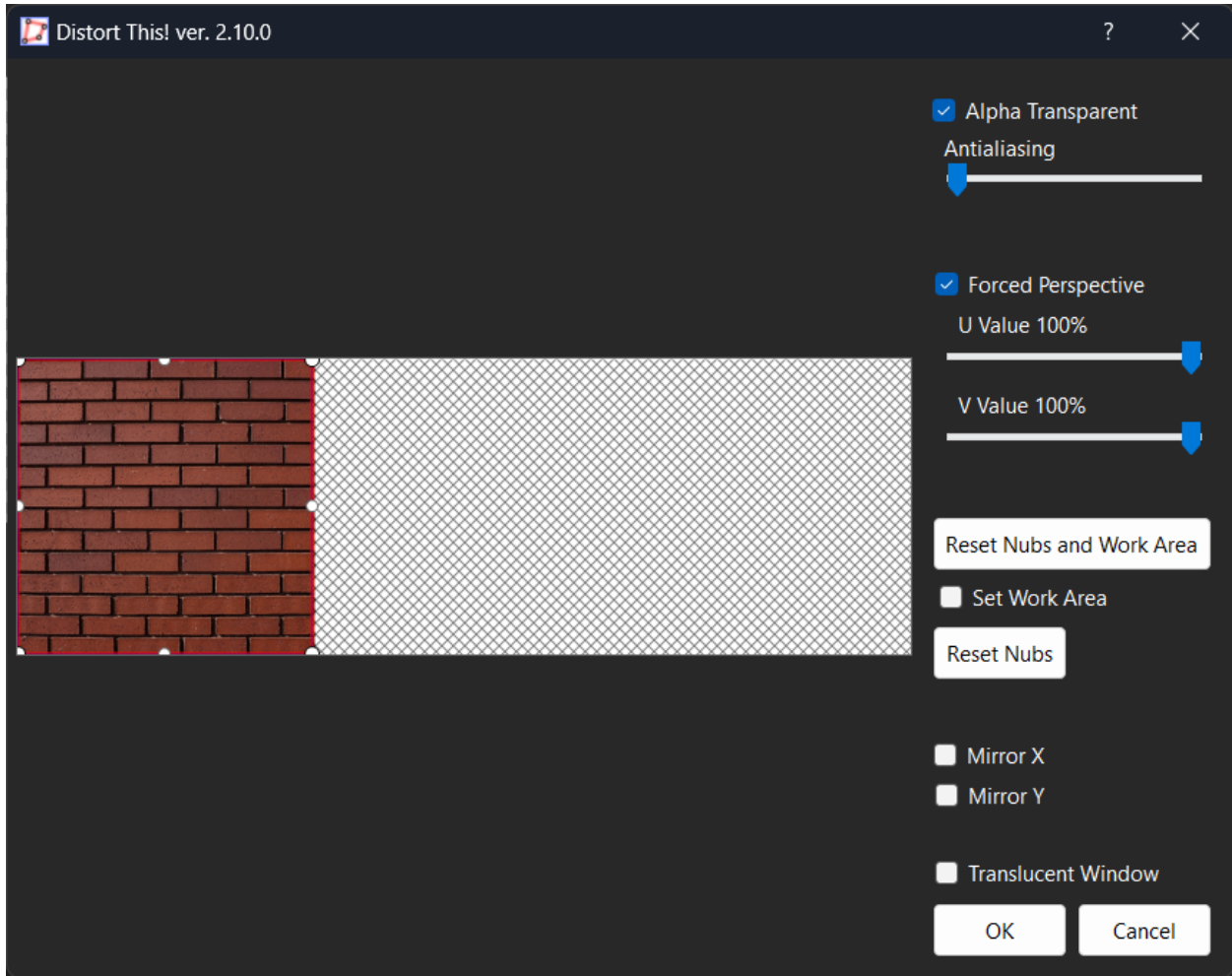
This will allow room for the distorted trapezoid on the canvas, otherwise an error message may prevent you from completing the operation.



Open the “Distort This!” plugin under “Distort” section of the “Effects” menu.
Ensure “Alpha Transparent” option is ticked.

On the right hand side panel, click “Reset Nubs and Work Area”.
This will reset the four distortion points to the corners of the canvas.
Only the confines of this rectangle will be distorted.

If you have a square canvas, these points will move to the exact corners of the texture.
However, if you have a rectangular canvas (as in the screenshot above), then you should tick
the “Set Work Area” box, and click and drag a rectangle around your texture, in the preview
window.



We will create the left hand side Side texture first, and later flip it horizontally to be saved as the right hand side texture.

For creating the 0L1S texture for resMode 3 (640x480), the line in reigons_etc is:

```
0L1S 0,-80,79,559 80,640 79,-1 79,480 79,79 79,560
```

The bounding rectangle of the texture is 80x640 pixels in size.

We are interested in the last two pts for correctly locating the compressed pts of the trapezoid shape. In this case 79,79 is the local point for the top compressed corner, relative to the top left corner of the image. 79,560 is the local point for the bottom compressed corner.

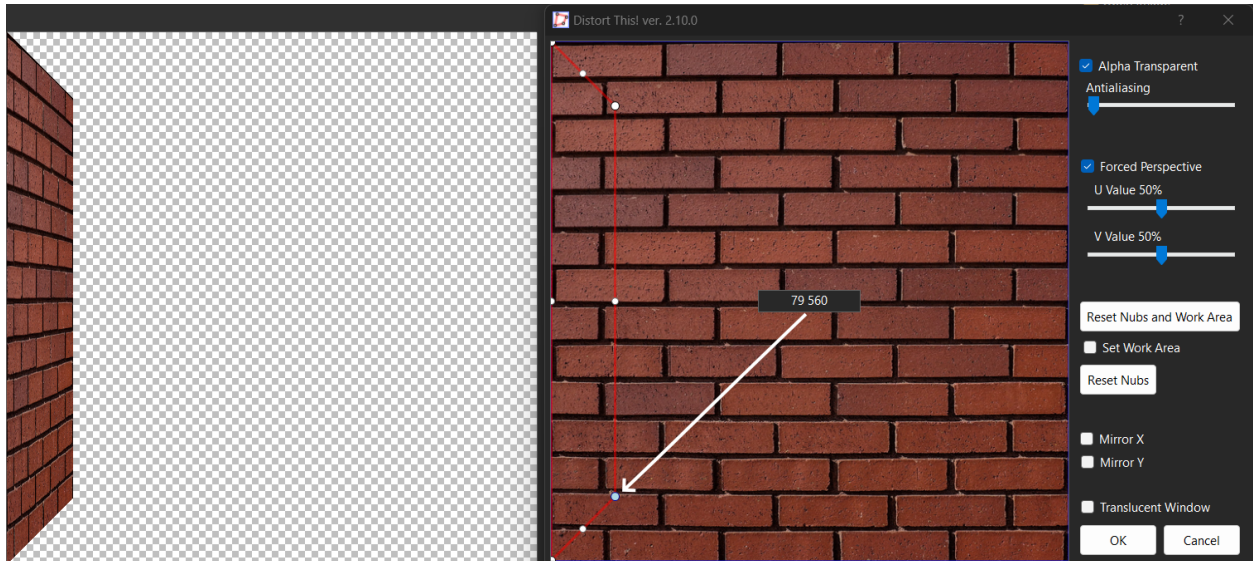
The pts are in x,y format, and 79 in this case means “pixel number 79” which equates to “the 80th pixel”, as paint.net begins counting pixels from the number 0.

In the “Distort This!” plugin window, double click on the white circle in the tight top right corner of the preview. A small text window will appear with the coordinates of this point.

Enter ”79 79” (with a space character separating the values) and hit enter. The point will now move to that coordinate.

Do the same with the bottom right circle, and the coordinate “79 560”

Setting the “Forced Perspective” sliders to 50% each, provides a natural perspective, preventing the image appearing skewed into the foreground, or background. The value strings may sometime say “100%” when the sliders are at 50%, but the live preview should indicate the correct setting.



Click OK to complete the distortion effect.

Now make a rectangular selection around the trapezoid, and crop to it.

Touch up the texture if necessary.

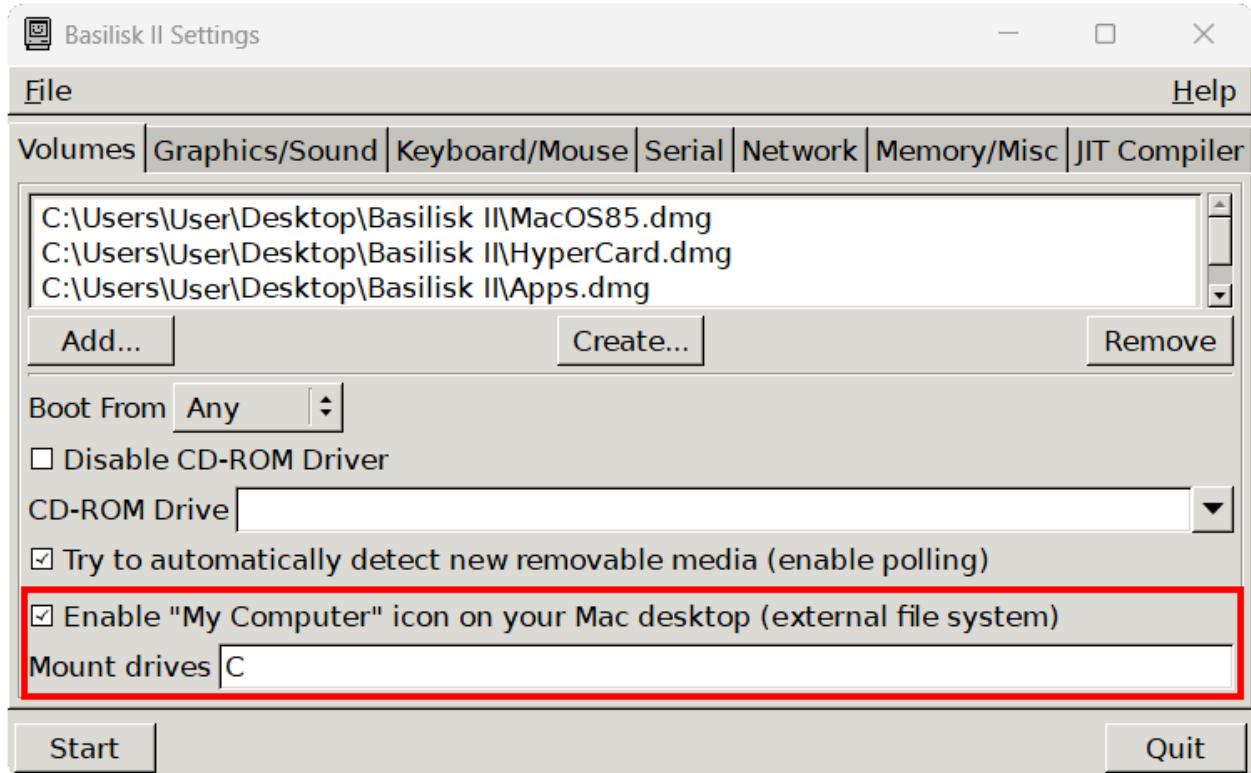
Remember to add a solid black line to the top and bottom slopes, within the image confines, for a crisp look, in-game.

Double check the image size reported at the bottom of your editor’s window. Ensure it matches the rect in your “regions_etc” line.



Export the image as a BMP. The top and bottom corners that were transparent will now be filled with opaque white pixels.

You can transfer your BMP files to your vintage Mac in several ways, but I recommend using Basilisk II's Shared Network Drive option to create a file portal between your host and emulated system.



On the Vintage Mac Side

The following steps may seem a bit arbitrary but they have been tested to provide the best results:

- Launch the QuickTime Picture Viewer, and open one of your BMP files, and copy it to the clipboard.
- Launch Photoshop. Adjust PhotoShop's preferences to enable "Export Clipboard", under Preferences > General > More.
- Create a New Photoshop Project. The Info dialog window should detect the resolution of the current contents of the clipboard. Proceed with the recommended canvas size and settings.
- Paste the contents of the clipboard onto the canvas.
- Deselect all pixels (Cmd + D)
- If your texture is B&W, then
 - Invert the Image (Cmd + I)
 - Invert a second time, to return to the correct polarity.
- Open the Stack in ResEdit (one way is to drag the stack icon onto the ResEdit icon).
- Navigate to PICT resources.

- Create a new resource and press Cmd + I to open the “Get Info” window.
- Tab once to move to the Resource Name field.

Enter the appropriate resource name and press Return.

(See the “[About Shared Textures](#)” chapter for resource naming convention).

Back in PhotoShop:

- Select all pixels (Cmd + A)
- Copy (Cmd + C)
- Paste into the PICT resource window.
(If you copy from PhotoShop before opening the stack in ResEdit, then the clipboard will lose it's contents).
- Check that the pasted image appears the same size in ResEdit as the original in PhotoShop.
- Close the resource window in ResEdit, and close your PhotoShop document.
- Close the BMP in QuickTime, and open the next one.
- Repeat

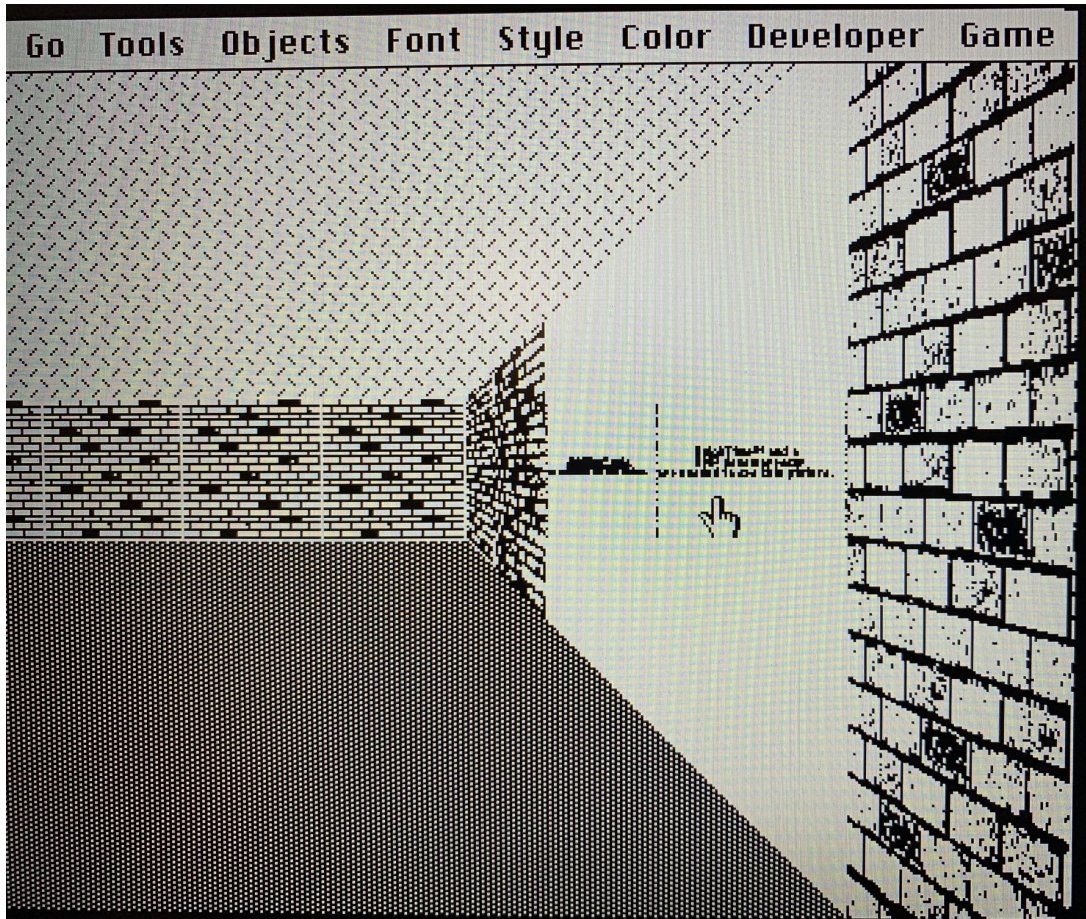
Yes, there are some obtuse methods in there, but following these steps will ensure the the final picture resource maintains it's resolution, is converted to native PICT format, and is the correct polarity (if black and white).

What Happens If I Do Things Differently?

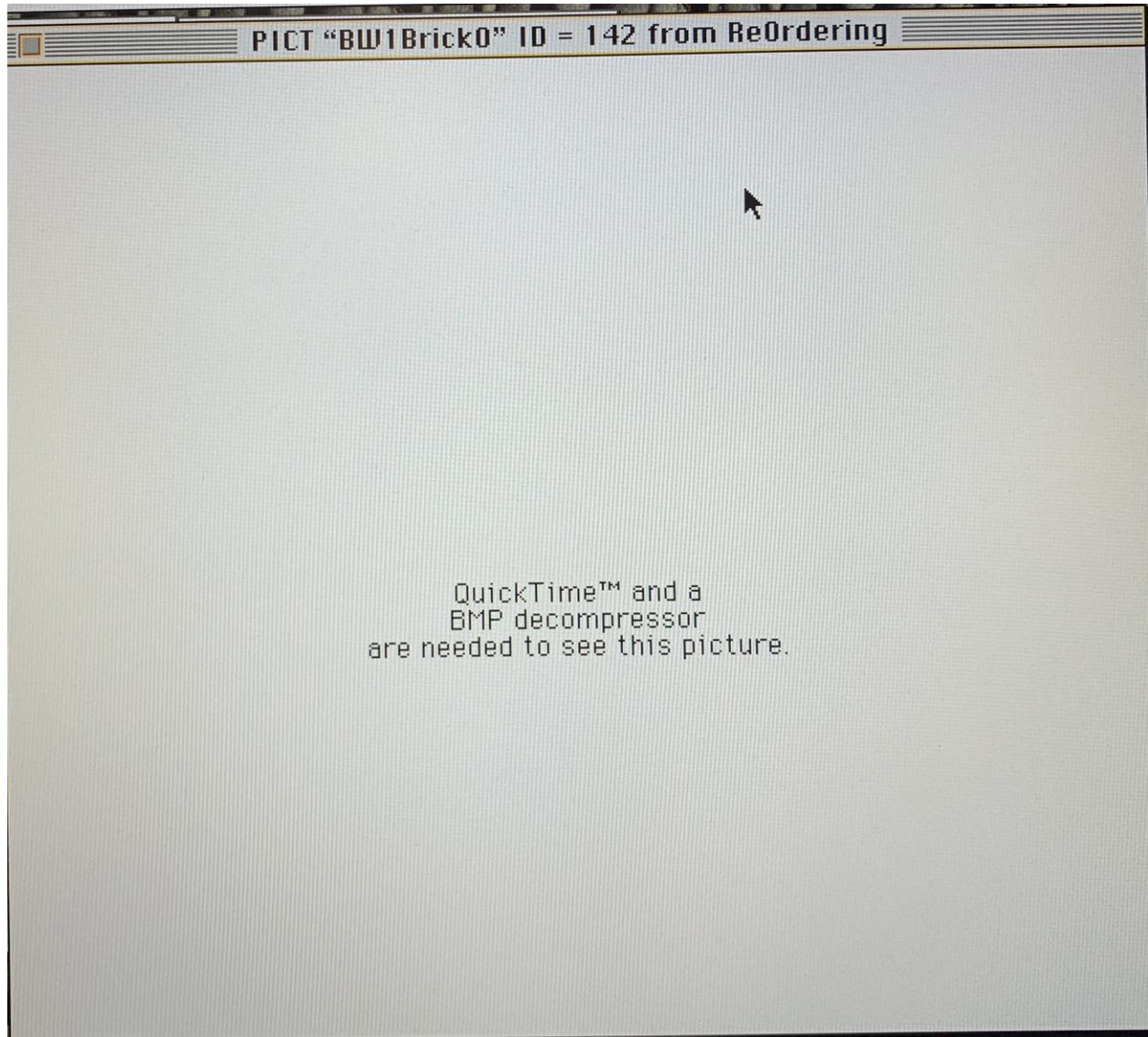
If you copy the image from the QuickTime BMP viewer and paste directly into ResEdit, it seems that the image data remains in BMP format, and does not get converted into PICT format, despite existing as a pict resource. When this occurs, there is no sign that anything is amiss, as the image is clearly visible as a PICT resource when viewed in RedEdit, as well as the HyperCard stack during gameplay.

However this changes when testing the stack on another startup disk that does not have QuickTime, and the BMP decompressor installed. In this scenario, all affected images are instead replaced by a rasterised error message.

When this was first discovered whilst testing on an emulated Macintosh II model at 640x480 resolution, the message/image was so reduced in size, that it was completely unintelligible and looked quite cryptic.



By testing the stack on a higher resolution system, this message was found to read:

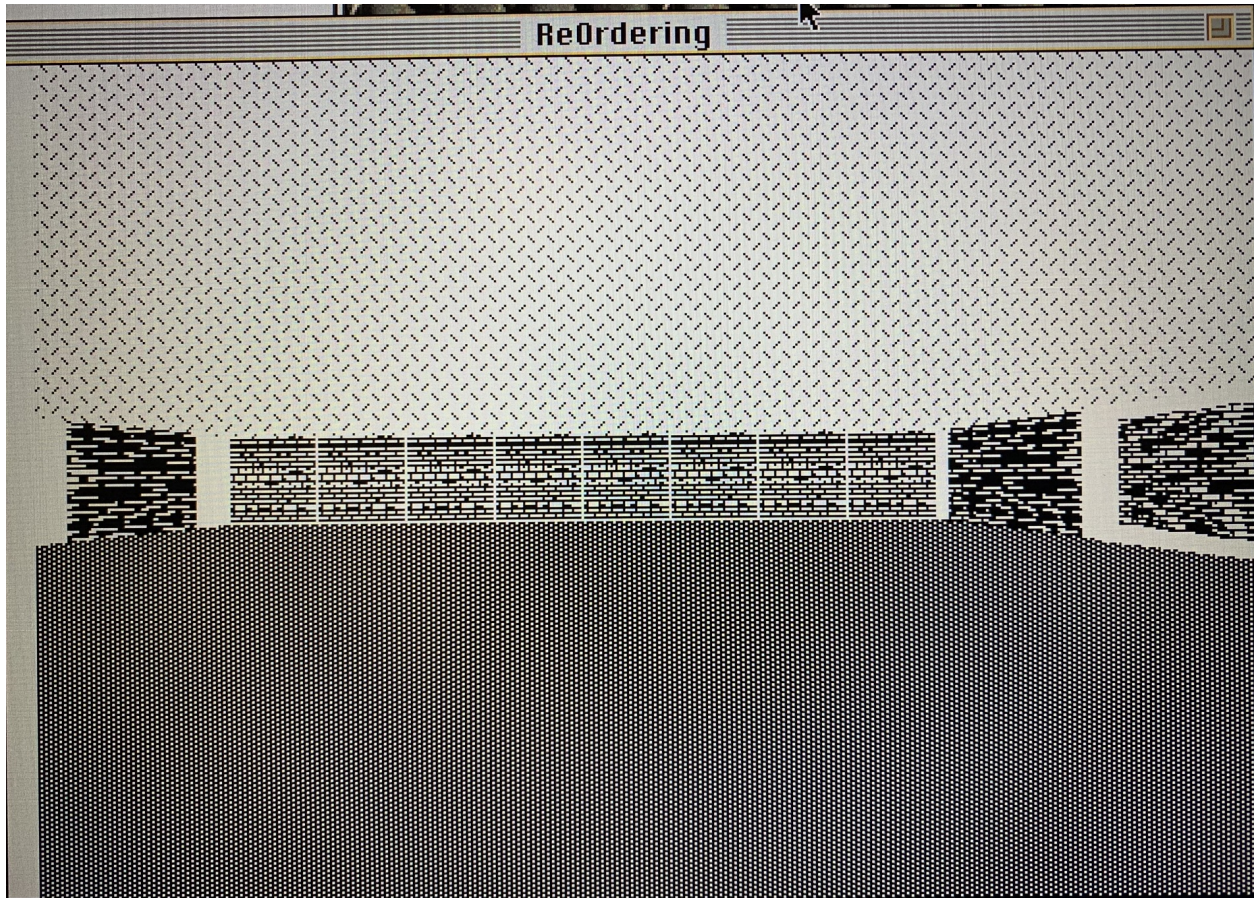


It is assumed that the resident BMP Decoder, loaded during startup, is called to decode images of this nature as they occur, while the application displaying the image is left unaware.

Pasting the image into PhotoShop temporarily, before copying and pasting again into ResEdit, appears to force the image data to convert from BMP, into raw, thus allowing the image to render on systems lacking a BMP decompressor (such as System 6.0.8).

So why bother with the Quicktime BMP Viewer at all, and not open the BMP file with Photoshop first?

If the BMP files are opened directly in PhotoShop (versions 2,3 and 4 tested), then copied and pasted into ResEdit, this will often result in a reduced size image arriving in ResEdit (about 75% the original image size). Most times this is immediately visible upon pasting into the PICT resource window. Other times it will appear as the full size image here, but alas shrunken when the image resource is drawn to screen during gameplay.

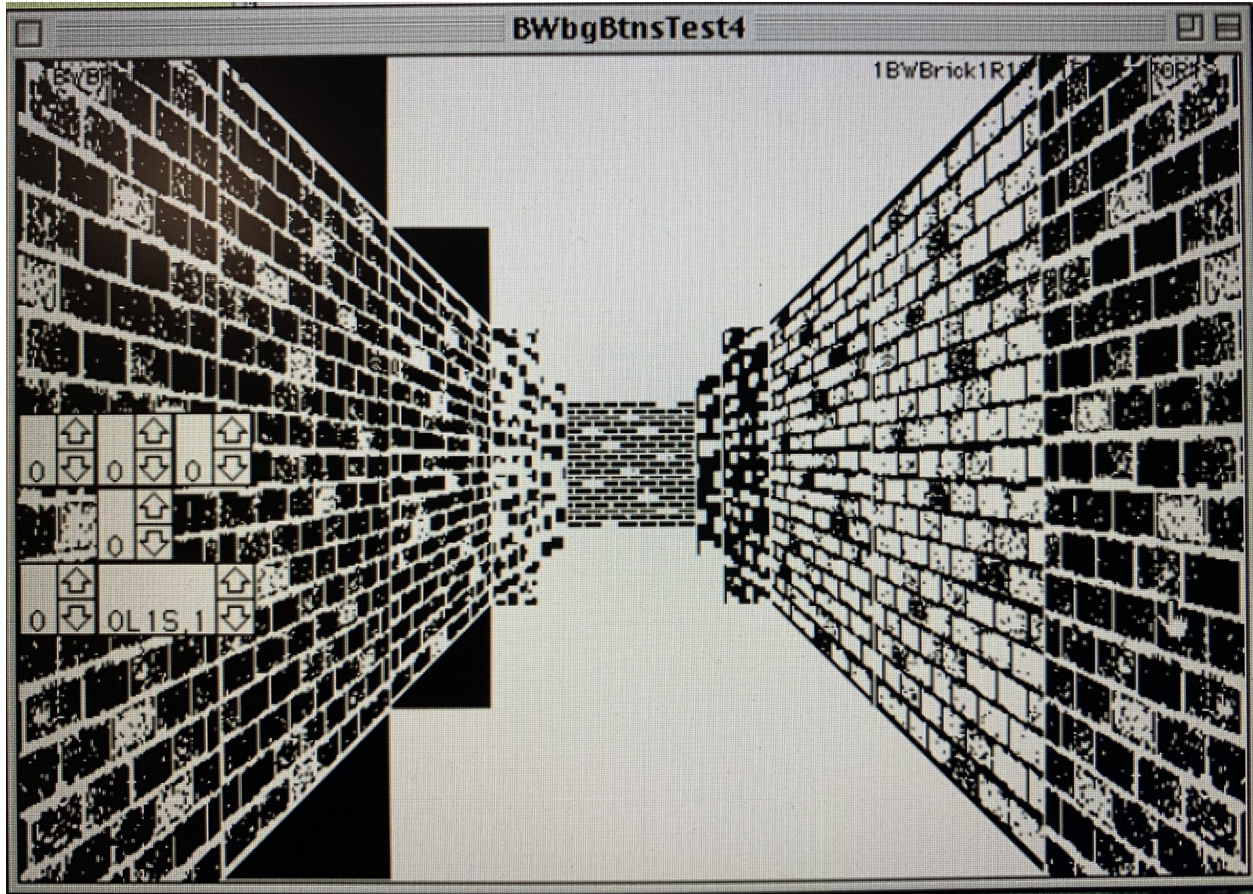


I can offer no explanation why, but the workflow of opening the BMP in Quicktime, pasting and copying through PhotoShop, before finally pasting into ResEdit provides the most reliable results.

But theres more.

If, when processing black and white graphics, the image polarity is not inverted twice, then another unexplained phenomenon occurs.

The B&W image exhibits the correct polarity when viewed in Photoshop, and in ResEdit, but inverted when viewed in HC.



When viewing the image in question with several different applications, it appeared that the issue was related to the age, or generation of the software products interpreting the image.

The following applications released ~1991 or later, displayed the image in the correct polarity.

- PhotoShop 2,3 and 4,
- The PICT Viewer function in the Hypercard Power Tools "Resource Mover" Stack.

Older applications displayed the image inverted:

- All HyperCard releases,
- MacPaint 2.0,
- Mac OS Scrapbook (Tested from System 6 to 8).

Inverting the image in PhotoShop, and then back again, seems to edit the image data in such a way as to remove this ambiguity that specific applications have when interpreting it.

Inverting the image (one operation), and then saving to a PICT resource, resulted in the image being drawn in HyperCard, inverted. This is the correct polarity for this case, because the image is meant to be inverted.

This indicates that inverting the image once is enough to solve the interpretation error, but then must be inverted a second time, to arrive back at the correct polarity.

The image which has had it's data corrected, is subsequently drawn with correct polarity in all other "older" apps.

It was found that copying from the QuickTime BMP Viewer, and pasting directly into an "old" app resulted in a truncated image being pasted (as well as inverted polarity, if the image was black and white).

These issues were replicated on a PowerPC G3 machine running Mac OS 9.2 (running many of the same 68K software binaries), indicating that the issues were not exclusive to the 68K emulation of Basilisk II.

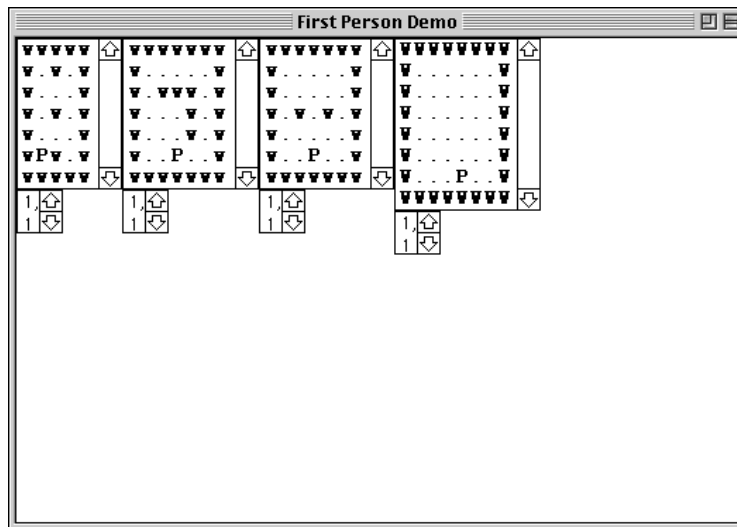
Another application, "GIF Converter" was tested, but was found to introduce noise into black and white images.

It was found that the MacPaint series was unable to open the BMP files created on the host machine.

How to Add & Edit Levels

To edit levels or add your own, open the stack, decline loading a level, and then enable Debug Mode from the Developer menu. Navigate to the "levels" card (the next card after "game").

Matching pairs of fields named "grid_X" and "gridcoords_textures_opacity_X" can be found here, where X indicates the number of that level.



The "grid_X" field contains the layout of the level from a top down perspective, with walls denoted by lower-case "w" characters, walkable empty floor space denoted by "." characters, and the initial Player Position denoted by a capital "P". Players are automatically set facing north (towards the top of the field) whenever a level is loaded.

The “gridcoords_textures_opacity_X” field specifies the textures assigned to each side (n,e,s,w) of each wall, present in a level. These fields have been compressed to the minimum size for maximum screen real-estate.

The “gridcoords_textures_opacity_X” fields follow the format:

word 1 - gridCoord (x,y)

word 2 - Name of Texture per side of wall (n,e,s,w)

word 3 - Opacity of Texture per side, (t)ransparent or (o)paque, (n,e,s,w)

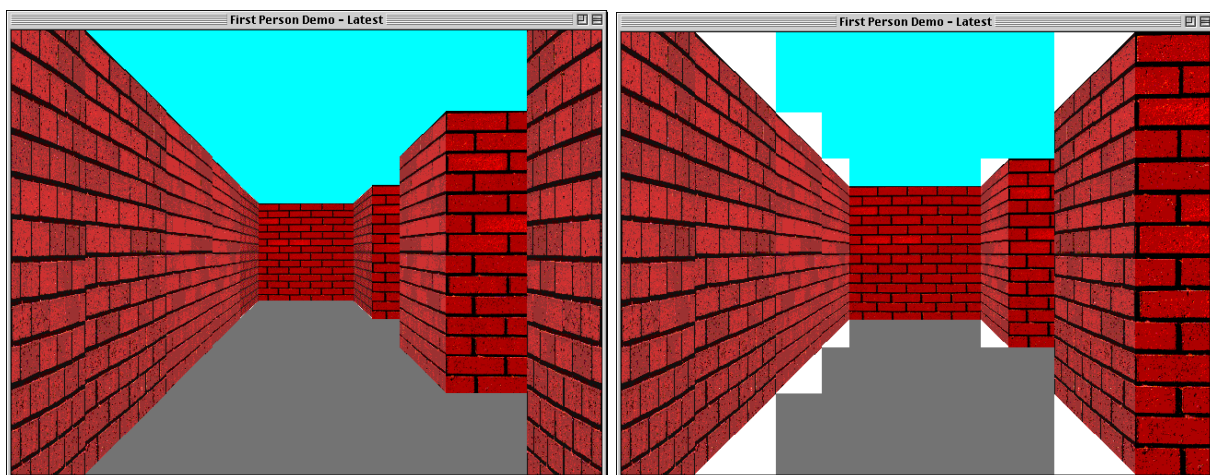
E.g.

- 5,4 Brick,Brick,Brick,Brick t,t,t,t

This example suggests that the Wall placed at character 5 of line 4 in the grid field, has the “Brick” texture assigned to each of its sides, and each of those textures is transparent.

The textures assigned to each side of a wall are listed from the north side of the wall (facing the top of the grid field), clockwise.

The Opacity option is a bit counter intuitive, in this release. Opacity / Transparency is not fully supported yet. Transparency is the default case, and assists the colour textures being drawn on screen. The void top and bottom corners of trapezoid textures of side regions, are solid white, and the transparent command is applied to those corners, allowing the ceiling and floor textures to appear correctly on screen, as seen in the screenshots below.



Left: Transparent setting.

Right: Opaque setting.

Make a New Level

First, lets increment the maximum number of levels.

On the “levels” card, click on the number in the field at the bottom right corner of the window, labelled “Number of Levels”

Press Command + A to select all text in this field, and replace it with a single integer of the new maximum number of levels you want.

Next, select the Field Tool from the Tools menu, and then with the Option key held down, click

and drag on one of the “grid_X” fields to make a duplicate (the text contents will not copy over with the new field).



Click and drag on the new field to reposition it somewhere on the card where it does not overlap any others.

Double click on the new field and change the number in its name to the new maximum number of levels.

Repeat these same steps for one of the “gridcoords_textures_opacity_X” fields.

Write your new level layout into the new “grid_X” field.

It is recommended to use a mono-spaced font, and enable the Bold and Extend options under the “Style” menu, to make the text easier to read.

Remember to fully enclose your levels with an uninterrupted boarder of walls, and place a capital “P” somewhere for the players starting position. There is currently no limit on map size, but only 8x8 rooms have been tested.

Next, go back to the “game” card, and load your new level from the “Game” menu.

After some thinking, there should be no graphics drawn on screen, only the developer objects should be visible. This is to be expected, as we have not assigned any texture information to your level yet.

Click on the “PopulateGridCoordsTexts” button toward the bottom right corner of the card.

The script in this button does the bulk of the work when assigning textures to the walls present in the grid. It will auto-generate the contents of the “gridcoords_textures_opacity” field underneath, with the required data, based on the current grid, and apply the “Brick” texture to all walls found. It is then possible to browse through the field and replace the stated textures with any other potential texture you want applied to walls.

If you have made additional textures, and want to add them to your level, it may be easiest to copy the contents of the “gridcoords_textures_opacity” field into a text editor to make any changes.

Once you are done, or if you are sticking with the single “Brick” texture, copy the text block, and paste it into your new “gridcoords_textures_opacity_X” field on the “levels” card.

The next time you load your new level, the scene should draw on screen.

Edit an Existing Level

To edit any of the included levels in this release, simply edit the text in one of the “grid_X” fields on the “levels” card. Make note of the number of the level you are editing. Follow the same Level design rules stated in the [“Make a New Level”](#) section, above.

Then navigate back to the “game” card, and load your level from the “Game” menu. The scene on screen is likely corrupted. This is to be expected, as the “gridcoords_textures_opacity” data doesn't match your new level layout.

Click on the “PopulateGridCoordsTexts” button toward the bottom right corner of the card. This will regenerate the contents of the “gridcoords_textures_opacity” field underneath.

If you have made additional textures, and want to add them to your level, it may be easiest to copy the new contents of the “gridcoords_textures_opacity” field into a text editor to make any changes.

Once you are done, or if you are sticking with the single “Brick” texture, copy the text block, and paste it into the “gridcoords_textures_opacity_X” field on the “levels” card, where X is the number of the level you have edited.

The next time you load your new level, the scene should draw on screen. Disable Debug Mode to hide all developer objects, and allow you to walk through your level.

Current Limitations

This release has several limitations:

The draw distance is limited to 6 gridCoords in front of player position. Regions in distance 7 and beyond will not be translated, and any walls found will not be drawn to screen, which in turn results in the ceiling and floor not being drawn above and below those regions, when set to B&W Mode.

As the draw distance is increased, more regions are available to be translated, increasing the workload (not to mention more PICT resources must be created for textures, increasing stack size). Considering that regions, and therefore textures, get progressively smaller until becoming 0 pixels in size at the horizon, a decision must be made regarding at what point the results stop being worth the extra work.

Only a single texture is present in this release; “Brick”. It is quite tedious creating textures and resources for all possible regions, but a single texture demonstrates the capability of the system for now.

A feature planned for a future release, is to include the regions and rects for far away distances, but beyond a certain distance, fill in those regions with a pattern or noise, instead of a dedicated texture resource. This saves developer time creating additional resources, and the cpu cost of drawing them to screen, while offering the player a visual clue or hint as to the scene in the far distance, which could improve their sense of space.

Future Plans

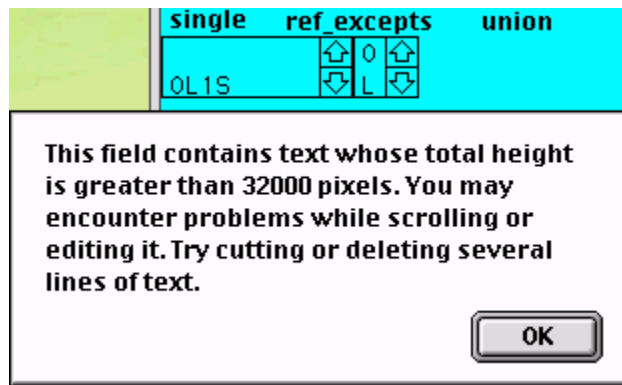
Priority features to focus on for the next release:

- Optimised reference data, sorted in such a way as to minimise cycles spent looping through lines.
- Optimised exception data, removal of duplicate entries and sorting to minimise cycles spent looping through lines.

- Revised method of calculating a scene, featuring a lean recursive handler.
- “Rogue-like” Mode.
- Investigate 3rd party external commands (XCMDs) to improve graphics drawing in HC.
- Additional textures.
- Optimised method of drawing far away regions
- Revised perspective equation that supports up to distance 9.

Discoveries & Trivia

With a `max_Distance` of 6, the “`ref_union_excepts`” field contains 2778 characters. When the field is shrunk down to its minimum size, clicking within the field with the cursor generates a warning:



Background objects are not addressed as such. They must be addressed as background objects of a card.

This doesn't work:

```
put fld "ref_union_excepts" of bg "640x480" into bg fld "ref_union_excepts"
```

This does work:

```
put bg fld "ref_union_excepts" of cd "640x480" into bg fld "ref_union_excepts"
```

Stacks are loaded into RAM in their entirety, including all resources for immediate access. Thus the size and quantity of resources could have a serious impact on the stacks ability to run on older machines.

It has proved difficult to replicate the Message Box functionality of placing text into a palette. Palettes are usually limited to a PICT resource, overlaid by a series of invisible buttons that send short messages. Perhaps there is a way to change the pointer of the PICT resource that the Palette loads, with custom text rasterised on it.

When specifying a rect for drawing a colour PICT resource to screen, the image will be drawn within the confines of this rect, but not on the pixels specified. This results in a 1 pixel border around the outside of the picture drawn on screen.

To alleviate this, each of the picture rect parameters (startx,starty,endx,endy) are adjusted outwards by 1 pixel.

A similar case occurs when drawing B&W graphics.

Each B&W PICT is drawn on a card button. The button rect must end 1 pixel further on both the x and y axis, in order for the full picture to be visible (endx and endy must be incremented by 1).

We only adjust these rects when drawing to the screen. The rects of the regions / textures are mathematically correct as they are.

HC does not offer the ability to edit the Apple menu, which is menu 1. It is able to edit the File menu (2) and onwards.

At many times during the program, we edit the lines in fields, whilst looping through those lines in a repeat statement. When deleting lines, we must take care not to disturb our line counter "curline", as we may delete lines "from under our feet", and discover that lines have been skipped over, or our pointer may point to a line, beyond the subsequent number of lines in the field, after making an edit.

During these cases, it is beneficial to use "repeat forever" instead of forcing our pointer to increment with every loop, which happens with "repeat with curline". This gives us greater control over when our line pointer increments, and of the conditions allowing us to exit the loop.

Variables in HC are volatile. Local variables are lost when the stack closes, and global variables are lost when the HC app closes.

Any data that should be saved between sessions should be written to a field during the closeStack handler, and restored during openStack.

"Deleting" a line is not the same as putting empty into it. The "delete" command removes final "return" character from the end of a line, where the "empty" command just removes the text, but preserves the line.

HC begins counting pixels from 0. This means that in cases where the screen resolution is 640x480, the pixels are addressed as 0,0,639,479. This affects the maths used when calculating pixels, e.g. halfWidth and halfHeight.

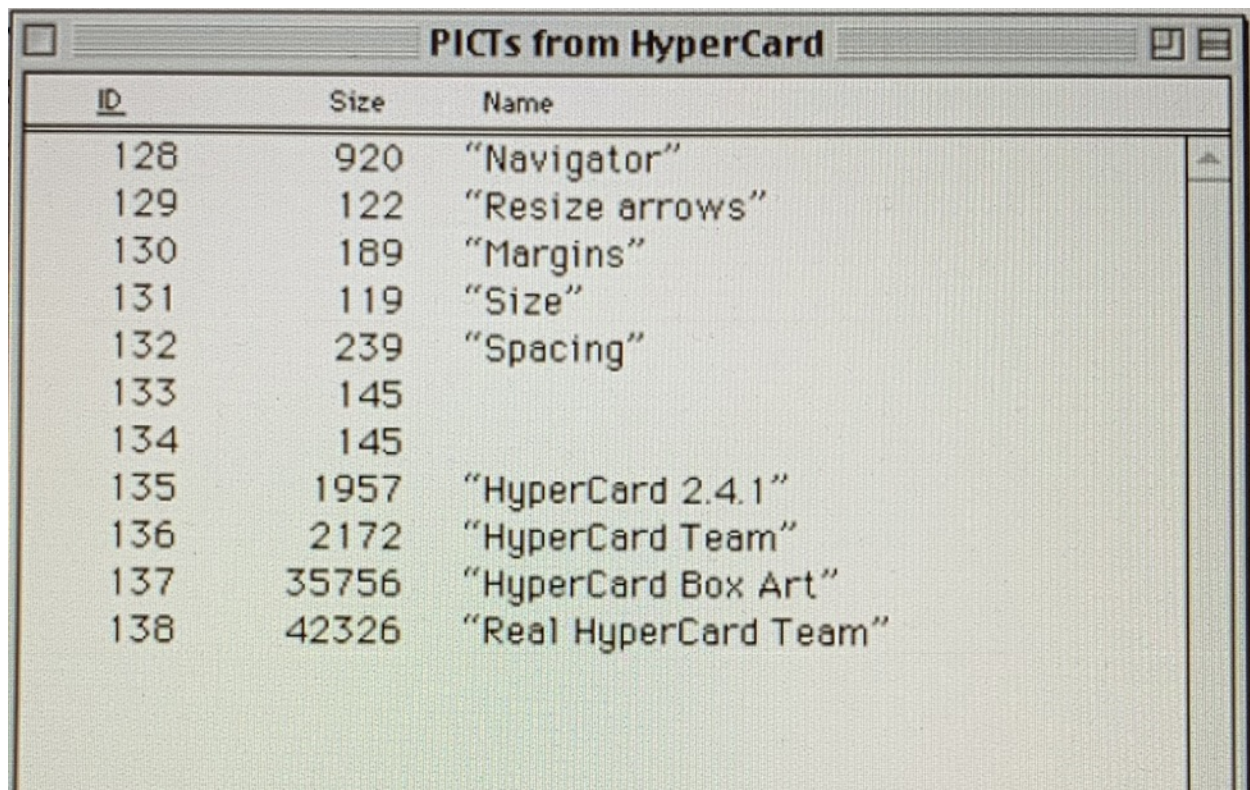
To determine the width and height of an object, we often subtract the start from the end, e.g. 10 to 19 is $(19-10) = 9$. But 10 to 19 inclusive is 10, and we must compensate for this.

Another problem is encountered whenever this maths is applied to a 0 value. e.g. 0 to 9 contains 10 possible values, but $(9-0) = 9$. This must also be compensated for.

It was discovered that HC does a poor job of solving exponential equations featuring a fractional or negative exponential e.g. $x^{0.8}$. The value of the exponent could be varied, but the resulting answer was often identical in many cases.

It seems HC is happiest with whole number exponents. For this reason, the Distance Units were calculated on a modern system, and the results were hard coded into the calcRegionRegularRects script.

The following PICT resource IDs are reserved by Hypercard:



ID	Size	Name
128	920	"Navigator"
129	122	"Resize arrows"
130	189	"Margins"
131	119	"Size"
132	239	"Spacing"
133	145	
134	145	
135	1957	"HyperCard 2.4.1"
136	2172	"HyperCard Team"
137	35756	"HyperCard Box Art"
138	42326	"Real HyperCard Team"

If a HyperCard stack features any PICT resources with these IDs, those custom resources will be loaded in place of the default resources, when called for.

For this reason, we have assigned our textures to IDs 142 and upwards.

PICT resource ID 25001 "NotEnufMemDisplay" is found to be present in stack files. This picture is displayed in place of any other PICT resources if the system has insufficient memory to display the resource.

Acknowledgements & References

Many Thanks to Ulrich Kusterer, whose video demonstrations have been invaluable in this

endeavour, and without their expert knowledge, this project would not have been worth starting.

Masters of the Void, & Kusterer, U. K., [Uli]. (2022, June 24).

Retro Programming: Building Myst with HyperCard. YouTube. Retrieved January 22, 2023, from <https://www.youtube.com/watch?v=QPmwnsqWVQY>

Uli's Home Page. (n.d.). <http://www.zathras.de/angelweb/home.htm>

The PICT image file format. (2017, January 15). Prepressure.

<https://www.prepressure.com/library/file-formats/pict>

Macintosh Paint: Summary from the Encyclopedia of Graphics File Formats. (n.d.).

<https://www.fileformat.info/format/macpaint/egff.htm>

HyperCard Tips and Tricks. (n.d.). HyperActive Software. Retrieved January 22, 2023,

from <https://www.hyperactivesw.com/hypercard/readertips/hctips.html>

HyperTalk Reference. (n.d.). The HyperCard Center. Retrieved January 22, 2023,

from <https://www.hypercard.center/HyperTalkReference>

Teach Yourself HyperCard. (n.d.). FolkStream. Retrieved January 22, 2023,

from <https://folkstream.com/muse/teachhc/index.html>

Resources. (n.d.). White Files. Retrieved January 22, 2023,

from <https://whitefiles.org/mac/pgs/t02.htm>

Tricks of the HyperTalk Masters | Scripting. (n.d.).

<http://www.jaedworks.com/hypercard/HT-Masters/scripting.html>

Textures:

Eliason, K. (2017, May 24). *Red Bricks Wall*. Unsplash. Retrieved January 22, 2023,

from <https://unsplash.com/photos/XEesx2NVpqWY>

Buda, M. (2019, March 23). *Gray Stone Wall Cladding*. Unsplash. Retrieved January

22, 2023, from https://unsplash.com/photos/UQ_0STgQpLY