GNT – GNT Is Not Tetris or the "Game that's Not Tetris"
A PixelStreams Implementation of Tetris

David Edell

Spring 2006

I. Introduction:

The goal was to create a fully functional Tetris game on the Celoxica Boards. The PixelStreams library is used as a basis for constructing the various game elements, and a description of the library itself can be found in the previous document. The Tetris game is fully playable, using a combination of the touch screen and buttons for input. Potential enhancements that can still be added include interchangeable background images, sound effects, and networked multiplayer capabilities, as well as making provisions for multi-colored blocks

II. Implementation & Discussion:

The Tetris implementation consists of several key parts: the user interface, the display filters, and the game logic.

The basic element of the game is the custp,PxsCust_TetBlock filter. This PixelStreams element accepts a block type and display location. The display location consists of the X and Y co-ordinates of the top-leftmost corner of the blocks grid. The Tetris block is represented as a 4x4 grid of cells, stored as a 16 bit unsigned integer. Each bit in the integer represents a specific section of the grid, in order from top-left to bottom-right. By default, the Tetris blocks have been defined as 16x16 squares.

In the original design, the TetBlock filter accepted as its input a type and rotation that together serve as an index to the stored blocks. The blocks, written as a macro expression, consists of four block representations for each of the 7 available Tetris blocks (4 were used during development of the filter). The block type is an index to the specific block currently active, and rotation is a 2-bit value that cycles between the four possible rotations of the block. The majority of the blocks in the Tetris game do not have unique configurations for all of their rotations. In designing the game under Handel-C however, it proves simpler and more efficient to hard-code each possible block configuration than

it would be to dynamically calculate the rotations. This simplification allows for the calculation of the block's display area to occur in 2-3 computational cycles with minimal hardware expenditure.

The final version of the game moves the index of possible block configurations into the main program. In its place, the TetBlock macro is simply passed the unsigned 16 representation of the active block directly. This modification allows for more flexibility in the usage of this filter, including the potential for easily adding additional blocks. An additional step may be required when changing the block type or rotation in the game's logic, however at the same time it reduces the number of lookups in the list required when performing game calculations.

All parameters to the TetBlock filter can be either constants or variables. In giving this (or other) filters a named variable, the filter can be modified during execution. In this case, the block's location and type are all variables that change throughout execution.

The TetGrid filter is responsible for displaying the game's board, specifically those blocks that have already fallen and been saved to the active game board. A standard Tetris game consists of approximately 10 columns and 20 rows. The grid in the game however contains 16 columns and 24 rows. The three columns on either side of the game's grid are deemed outside of the playable area, however are always initialized in the grid as filled. The default values for a row are stored in a macro expression called defRow. This buffer zone allows for simplified comparisons when computing if an active block may be moved, or if it must be saved to the current location. The bottom three rows are likewise reserved as a lower buffer, by default initialized to bufRow. Placing these filled regions outside of the viewable gaming area allows for the game's logic to freely and accurately compute the block movement for variable block configurations without checking for out of bounds errors in the grid.

The TetGrid filter requires an mpram of a specific type. It consists of an array of 24 unsigned 16-bit numbers. Each 16-bit number corresponds to a row on the display grid. The filter accesses this variable as read only memory defined with the name 'disp.' The game's main logic accesses the second port of the mpram as a ram access, allowing it to both read and write to the game grid to update and calculate game status. During

calculations, each row of the grid is compared to a row in the active block. The four bits of each row in the active block are selected separately from the activeBlock variable, and then padded back into a 16-bit unsigned integer. This value is then right shifted by a number of bits equal to the current column of the active block's leftmost side.

Bitwise comparisons are used to check if a move is valid. Before a block continues moving down vertically, a rotation is accepted, or horizontal movement is made, the game first checks if the new location of the active block is occupied. If it is, the block movement is disallowed. In the case of vertical movement, the block is saved to the grid when it can no longer move down. Horizontal movement comparison is based on the current row, however due to the method used this check does not prevent blocks from overlapping in the display. This can happen when a block on row x is occupied, but it is free on the row below it. Once the block starts moving into its next position, the movement is checked against that new row. Preventing block movement in these cases would interfere with the proper game play, however allowing it gives the illusion of two blocks moving into each other. One potential way to fix this with the methodology being used would be to introduce border regions between the blocks, which would also serve a double purpose of improving the game's appearance.

The game accepts a variety of user input (described in the next section). This input can activate one of several game modes or options. The game's status is stored in a 'mode' variable, detailed in the code comments. The game status will be modified in response to user input, and reset in the thread of execution handling the game logic. The modes serve as channels of communication indicating new or paused game status, as well as providing for block rotation and horizontal movement. An additional 'turbo' flag is available that allows a block to immediately drop to the bottom of the screen. Activating this flag causes the game to skip the VGASync between computations until the active block is reset (reaches the bottom).

The active game blocks on the screen are selected through the usage of several values. A random number generator continuously increments a variable whose size corresponds to the number of available Tetris blocks to choose from. This value is copied into the type variable when needed for both the active block (currently falling in the game grid) and the next block. A preview is shown on the top left of the display of

the next block to appear. An additional 2-bit rotation variable (initialized to 0) selects the sub-type of the block, cycling between the 4 possible orientations. Each time the rotation is changed, the activeBlock variable is reset from the rom using the current index. When it is time for a new block to be generated, the active block is given the value of the nextBlock. A new nextBlock is then computed by setting its type equal to the random number generator.

III. Gameplay

The goal in Tetris is to fit falling blocks together to fill an entire row. Doing so, will 'clear' that row, and shift all blocks above it down one row. Points are awarded for blocks dropped and rows cleared, with bonuses given for clearing multiple rows (also referred to as lines) at a time.

The scoring in this implementation is determined by macro constants at the start of the program. By default, 2 points are rewarded for each block that falls, and 8 points for each line cleared. An additional 8 points accumulates for each line when more than two are cleared at one time. The game's score is displayed on the top of the game grid.

Levels in a Tetris game are based on the speed of the falling blocks. In this version, the level is advanced with every 16 rows cleared. Level speed is determined by the interval, or number of VGASync's, before the block's vertical position is incremented. After a certain level, this value is increased, and a second level variable is incremented controlling the number of pixels with which the block moves with each update.

The Touchscreen provides the primary controls for this game. The display is currently labeled appropriately on where to touch to rotate a block, or move it to the left or right by one column. Pressing both buttons at once will result in a new game. The first button alone serves as a turbo button, causing the active block to immediately drop. The second button is used to allow the game to be paused, or resumed. Current game status is displayed on top of the game board, above the score.