2D Game Programming

- On Older Hardware
  - Computers were limited by slow CPUs and small memory sizes
    - Ex) 8-bit processor, 48KB RAM
  - Many systems did not have "secondary storage"
    - Ex) program loader (tapes, cartridges)
  - No floating point unit
- 2D technologies
  - Still remain today
    - Ex) Handhelds, game-capable telephones, interactive television

Data Structures for 2D Games

- Three key elements of classic 2D games
  - A way to encode the character graphics
  - A way to encode background images
  - A way to store the game map

Sprite-Based Characters

- Cel animation
  - Each frame of each character was painted on cellophane sheets with transparent area
- 2D games
  - Sprite
    - Store each character in a rectangular, bitmapped graphic
      - ✓ Detect transparent areas
      - ✓ The characters blend seamlessly with the BGs

How to encode the character graphics

Sprites

- Games that we know and love
  - Small graphics that are copied into memory to put on screen
- Xevious, Time Pilot

Sprite-Based Characters

- black and white
  - Store the sprites in black and white (or any two given colors)
  - Ex) 8x8 sprites(8 Bytes), The FG and BG colors are selected from a 16 color palette (1Byte: 4 bits+extra) ➔ 9 Bytes per sprite

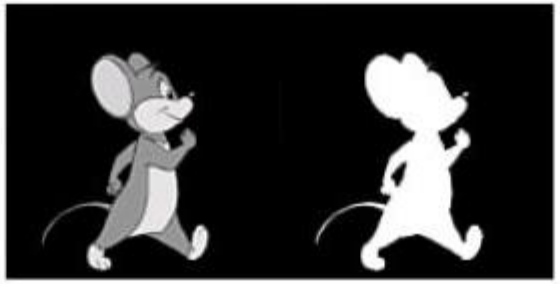
    The frame buffer: 256x176 pixels ➔ 32 x 22 Sprites ➔ approximately 6 KB (32x22x9 ➔ 6,336 Byte)
- 16 colors
  - Each pixel can be coded to 4 bits
  - Around 4 times more space, represent much richer scenes
  - Ex) 8x8 sprites ➔ 32 Bytes per sprite (=8x8x4 bits)

    The frame buffer: 256x176 pixels ➔ take up 23 KB (32x22x32 ➔ 22,528 Byte)

Sprite-Based Characters

- 256 color per pixel
  - These colors usually come from
    - ✓ A fixed palette or a palette freely defined by the user
  - Ex) 8x8 sprites ➔ 64 Bytes per sprite (=8x8x8 bits)
    The frame buffer: 256x176 pixels ➔ take up 46 KB
    the palette table: 256 colors of 24bits(RGB) each
    ➔ 3Byte(RGB:24bits) x 256 color = 768 Byte
- High-color sprites (16 bits per pixel)
  - Two option
    - ✓ Encode using 5-5-5-1
      - – 5 bytes ➔ red, green, blue 1 byte ➔ alpha
    - ✓ Encode using 6-5-5
      - – Encode the transparency color as one of the color combinations (a chroma key approach)
- True-color sprites (24 or 32 bits)
  - Each pixel: one double word(32 bits)
    - ✓ R(8), G(8), B(8), A(8)

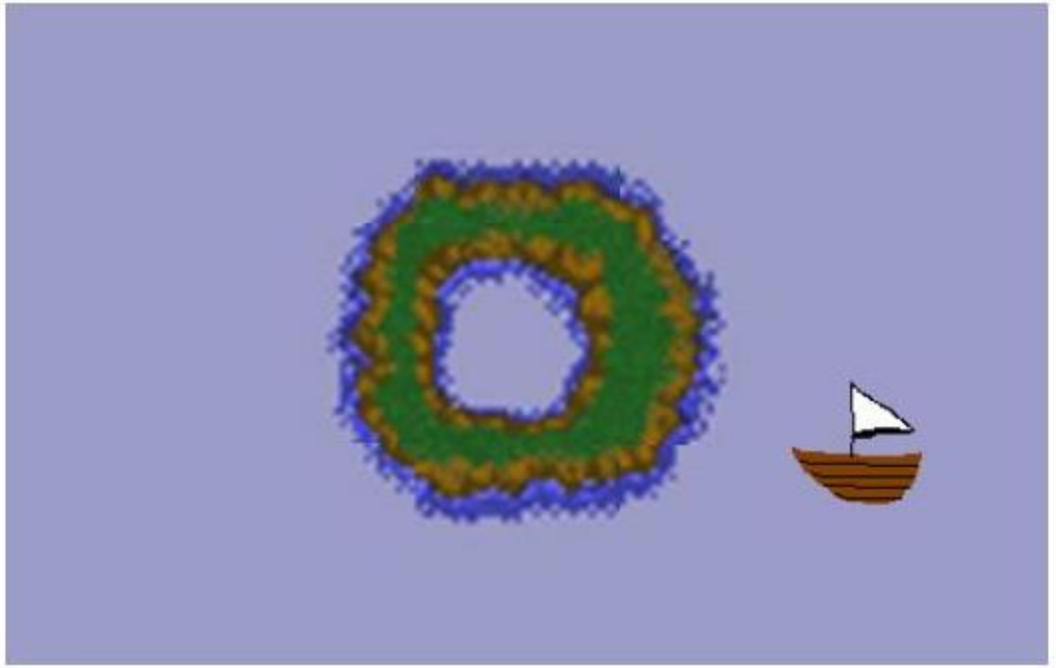
Sprite-Based Characters

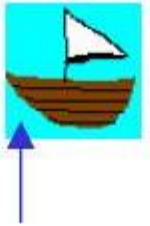


- **Transparency**
  - Masking approach
    - Use a separate 1-bit mask to encode transparent zones
    - Simple to code, but take a significant amount of memory
      - ✓ Ex) a 32x32, 256 color sprite ➔ 1 KB (32x32x8 bits)
        the mask ➔ 128 extra bytes (32x32x1 bits ➔ 1024 bits)
  - Alternative technique
    - Reserve one color in our palette as the “transparent color”
    - Lower-color platform ➔ the loss of the one color might be unacceptable
- **Blitting**
  - Layering sprites onscreen (25 screen updates per second)
  - “blit” ➔ “block image transfer”

Color Key

- Color to represent transparency
  - when a tile or sprite is drawn, pixels with the color key are ignored
  - Why?
    - because those pixels should not be drawn



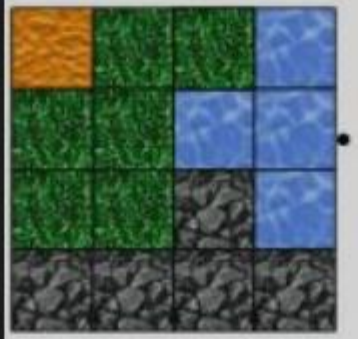


Specify this precise shade of blue as color key when:

- reading file
- drawing object

Mapping Matrices

- **Mapping**
  - Compression technique to make data fit on the very small memory chips
    - divide our game world into a set of tiles
      - ✓ Each tile represent a rectangular pattern
    - no mapping) level: 5x5 screens, screen: 256x200 pixels, palletized to 16 colors → take up 1.25MB



1	2	2	3
2	2	3	3
2	2	4	3
4	4	4	4

- Mapping) 256 different tiles, tile: 8x8 pixels
  each tile → 8x8 = 64 = 32 B (using 16 colors)
  tile list → 32x256 = 8 KB
  the size of mapping matrix → 160x125 (256x5/8, 200x5/8)
  whole table → 160x125 = 20000 B (256 possible tiles)
  total → tile list + whole table → 27.5KB
  (매핑하지 않은 방법에 비해 50 배 차이)

Ex) Mario Bros, Zelda, 1942…

Mapping Matrices

- **Tile Tables**
  - A list of background images that can be tiled and combined using a mapping matrix to create a complete game map
  - Format of Tiles
    - Tile size
      - ✓ Used to be powers of 2 ➔ Increased efficiency
        - – blitting routines ➔ using words (32bits value)
      - ✓ Whether all tiles will be the same size or not
        - – Classic games: equal-size tiles for easier screen rendering
        - – RTS using isometric view: different tile sizes
    - The color format of the tiles
      - ✓ More colors the better, but more memory, more bus usage, less performance

How to encode background images

Mapping Matrices

- **Memory size of a single tile**

  Size = bits per pixel * wide * tall

- **Number of Tiles**
  - **More tiles means nicer graphics, more memory**
  - **Ex) 256 tiles ➔ unsigned, 8 bit number**

    300 tiles

    ✓ **Using 9 bit ➔ 512 values, but hard to access**

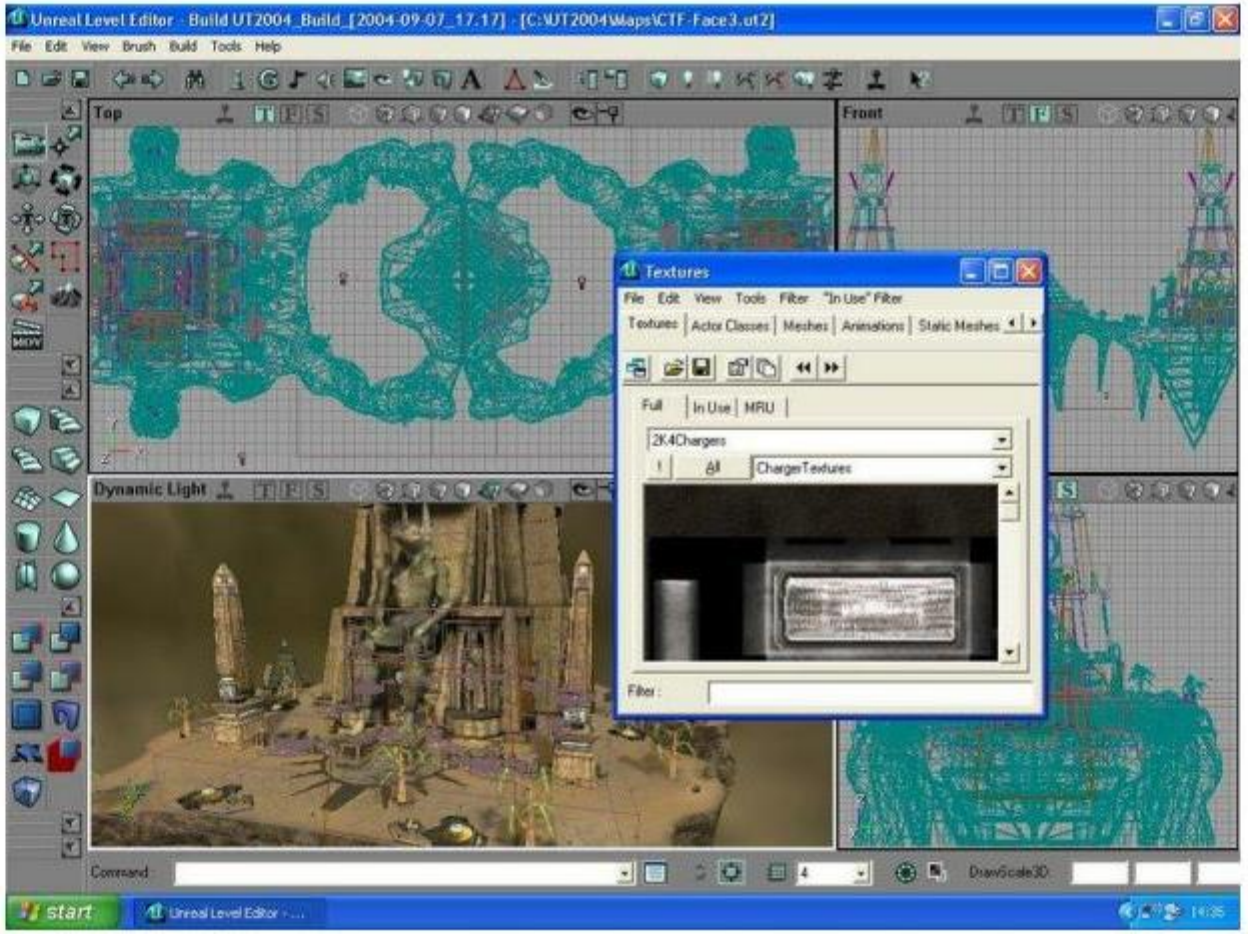
    ✓ **Using a 16-bit value ➔ take up double the memory, give simple access**

What is a tile (generally speaking)?

- A building block of a game board
- Piece together tiles to create a world
- Why use tiles?
  - to conserve memory
  - graphics reuse
  - dynamic content

Why else is graphics reuse important?

- Because artist time is expensive
- Level designers can layout a map



How can tiles be dynamic?

- Random map generator
  - adds to game re-playability
  - a different game each time you play it

Identify tiles needed

- **Terrain**
  - **grass, dirt, sand, snow, water, mountains, etc.**
- **Walls**
- **Roads**
- **Buildings**
- **etc.**

- **And don't forget *terrain borders*. What's that?**