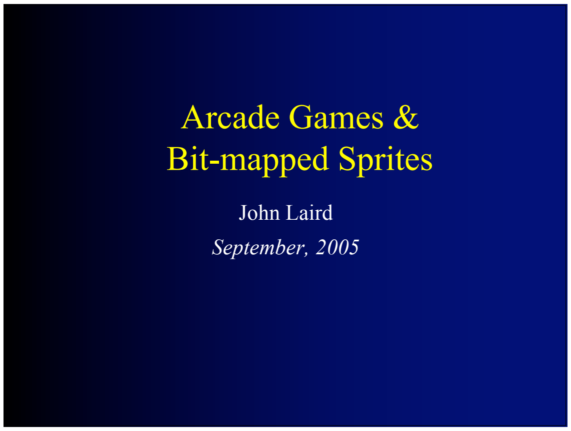
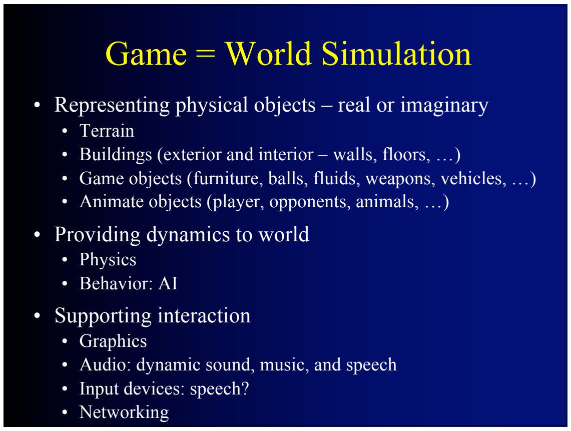


Arcade Games & Bit-mapped Sprites

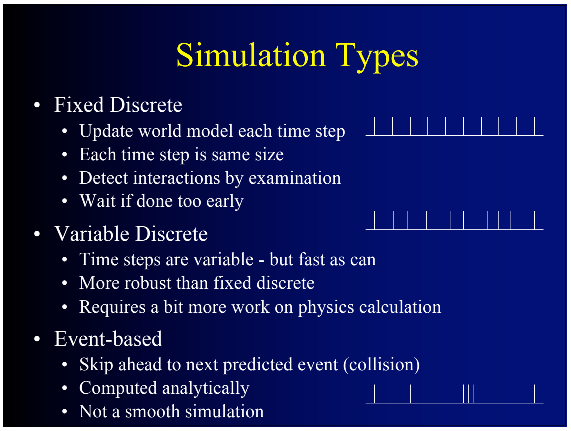
John Laird

September, 2005



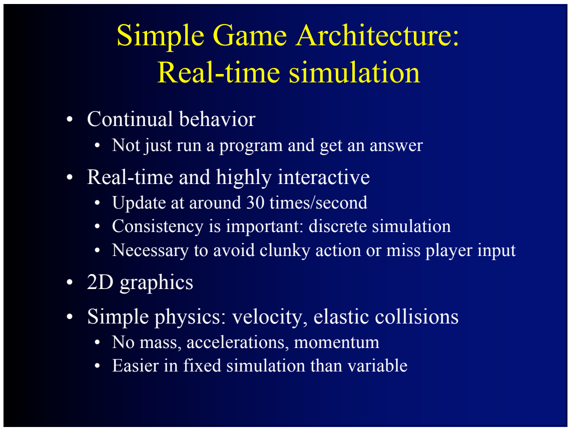
Game = World Simulation

- Representing physical objects – real or imaginary
  - Terrain
  - Buildings (exterior and interior – walls, floors, …)
  - Game objects (furniture, balls, fluids, weapons, vehicles, …)
  - Animate objects (player, opponents, animals, …)
- Providing dynamics to world
  - Physics
  - Behavior: AI
- Supporting interaction
  - Graphics
  - Audio: dynamic sound, music, and speech
  - Input devices: speech?
  - Networking



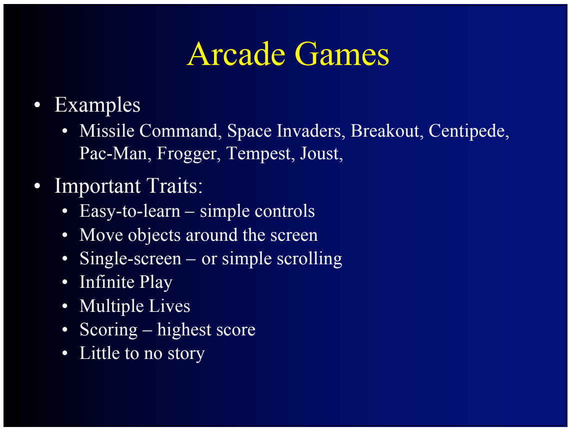
Simulation Types

- Fixed Discrete
  - Update world model each time step
  - Each time step is same size
  - Detect interactions by examination
  - Wait if done too early
- Variable Discrete
  - Time steps are variable - but fast as can
  - More robust than fixed discrete
  - Requires a bit more work on physics calculation
- Event-based
  - Skip ahead to next predicted event (collision)
  - Computed analytically
  - Not a smooth simulation



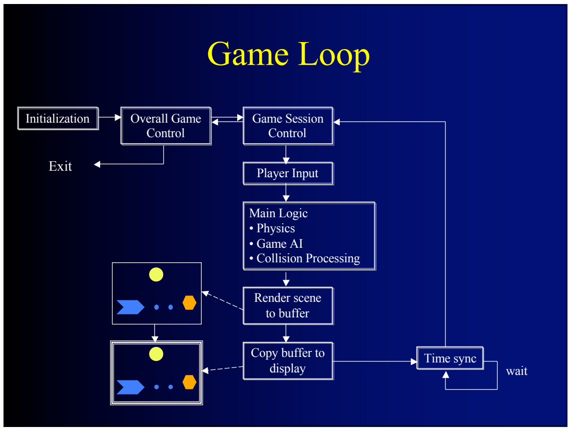
Simple Game Architecture: Real-time simulation

- Continual behavior
  - Not just run a program and get an answer
- Real-time and highly interactive
  - Update at around 30 times/second
  - Consistency is important: discrete simulation
  - Necessary to avoid clunky action or miss player input
- 2D graphics
- Simple physics: velocity, elastic collisions
  - No mass, accelerations, momentum
  - Easier in fixed simulation than variable



Arcade Games

- Examples
  - Missile Command, Space Invaders, Breakout, Centipede, Pac-Man, Frogger, Tempest, Joust,
- Important Traits:
  - Easy-to-learn – simple controls
  - Move objects around the screen
  - Single-screen – or simple scrolling
  - Infinite Play
  - Multiple Lives
  - Scoring – highest score
  - Little to no story



Game Loop

Initialization

Overall Game Control

Game Session Control

Exit

Player Input

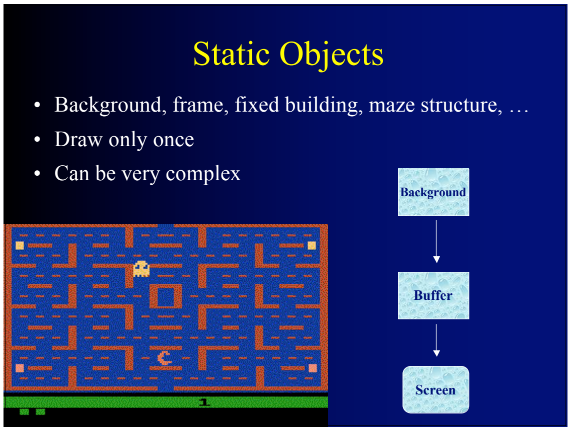
Main Logic
• Physics
• Game AI
• Collision Processing

Render scene to buffer

Copy buffer to display

Time sync

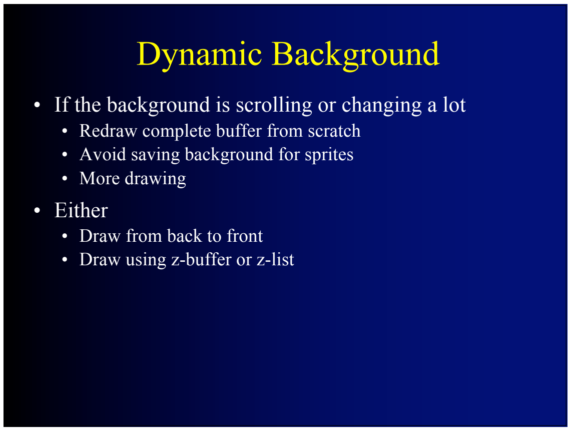
wait



Static Objects

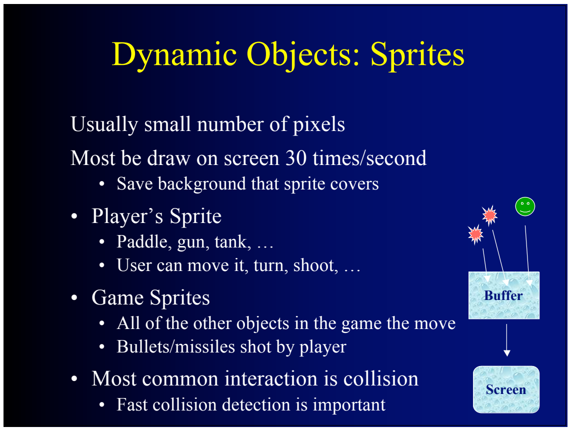
- Background, frame, fixed building, maze structure, …
- Draw only once
- Can be very complex

Background
Buffer
Screen



Dynamic Background

- If the background is scrolling or changing a lot
  - Redraw complete buffer from scratch
  - Avoid saving background for sprites
  - More drawing
- Either
  - Draw from back to front
  - Draw using z-buffer or z-list



Dynamic Objects: Sprites

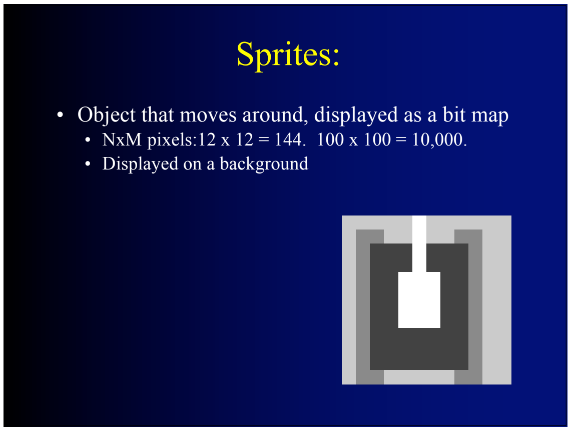
Usually small number of pixels

Most be draw on screen 30 times/second

- Save background that sprite covers

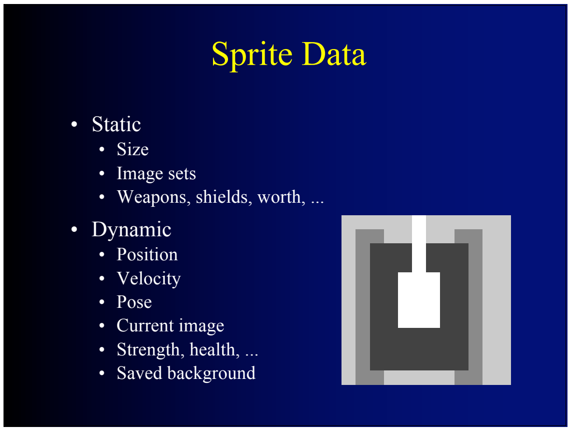
- Player's Sprite
  - Paddle, gun, tank, …
  - User can move it, turn, shoot, …
- Game Sprites
  - All of the other objects in the game the move
  - Bullets/missiles shot by player
- Most common interaction is collision
  - Fast collision detection is important

Buffer
Screen



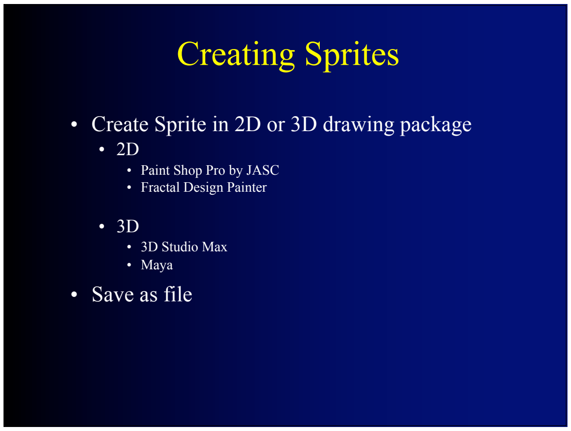
Sprites:

- Object that moves around, displayed as a bit map
  - NxM pixels:12 x 12 = 144. 100 x 100 = 10,000.
  - Displayed on a background



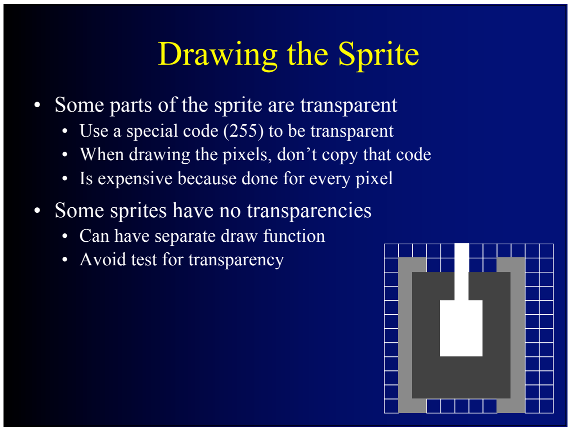
Sprite Data

- Static
  - Size
  - Image sets
  - Weapons, shields, worth, ...
- Dynamic
  - Position
  - Velocity
  - Pose
  - Current image
  - Strength, health, ...
  - Saved background



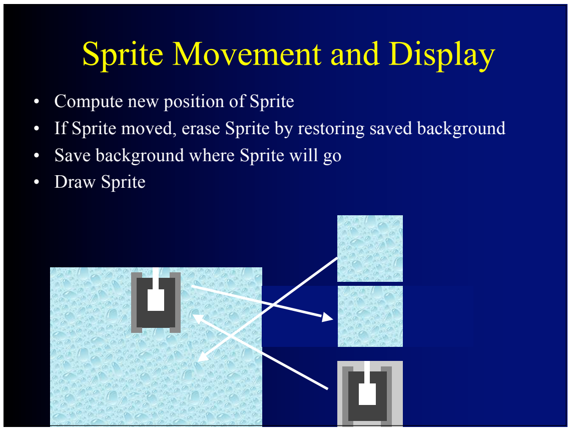
Creating Sprites

- Create Sprite in 2D or 3D drawing package
  - 2D
    - Paint Shop Pro by JASC
    - Fractal Design Painter
  - 3D
    - 3D Studio Max
    - Maya
- Save as file



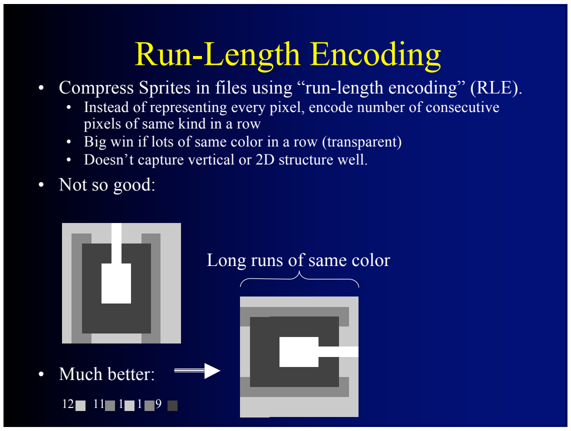
Drawing the Sprite

- Some parts of the sprite are transparent
  - Use a special code (255) to be transparent
  - When drawing the pixels, don't copy that code
  - Is expensive because done for every pixel
- Some sprites have no transparencies
  - Can have separate draw function
  - Avoid test for transparency



Sprite Movement and Display

- Compute new position of Sprite
- If Sprite moved, erase Sprite by restoring saved background
- Save background where Sprite will go
- Draw Sprite



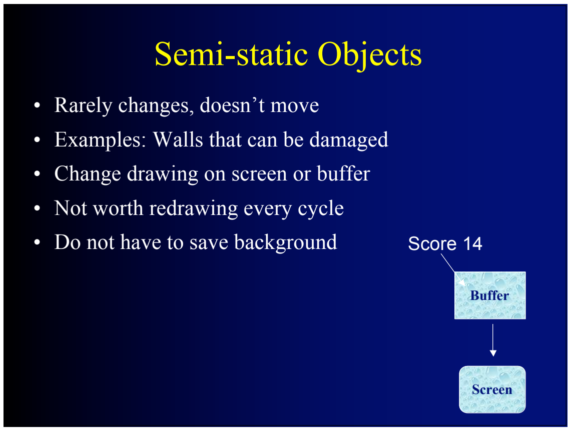
Run-Length Encoding

- Compress Sprites in files using "run-length encoding" (RLE).
  - Instead of representing every pixel, encode number of consecutive pixels of same kind in a row
  - Big win if lots of same color in a row (transparent)
  - Doesn't capture vertical or 2D structure well.
- Not so good:

Long runs of same color

- Much better:

12□ 11▨ 1□ 1▨9 ■



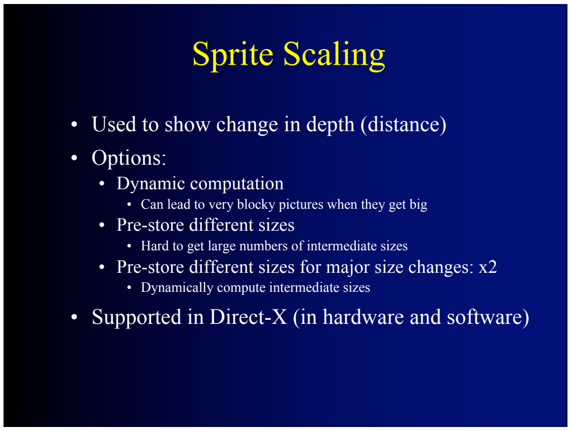
Semi-static Objects

- Rarely changes, doesn't move
- Examples: Walls that can be damaged
- Change drawing on screen or buffer
- Not worth redrawing every cycle
- Do not have to save background

Score 14

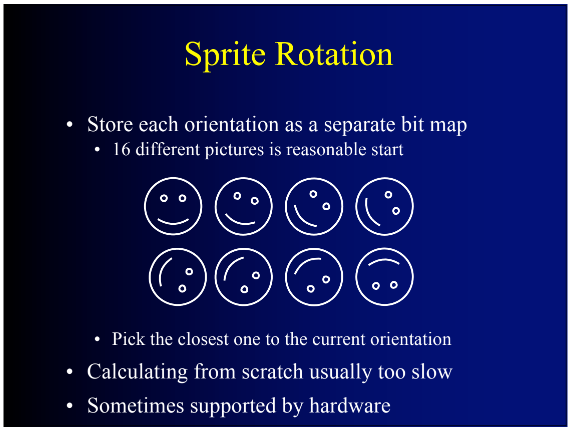
Buffer

Screen



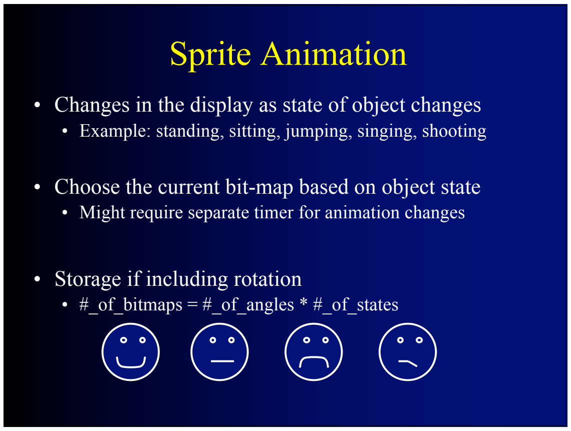
Sprite Scaling

- Used to show change in depth (distance)
- Options:
  - Dynamic computation
    - Can lead to very blocky pictures when they get big
  - Pre-store different sizes
    - Hard to get large numbers of intermediate sizes
  - Pre-store different sizes for major size changes: x2
    - Dynamically compute intermediate sizes
- Supported in Direct-X (in hardware and software)



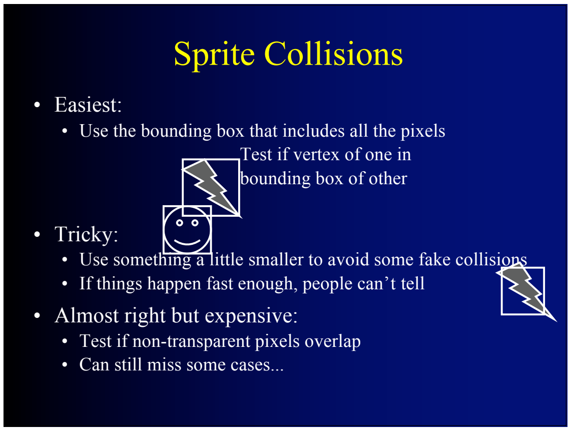
Sprite Rotation

- Store each orientation as a separate bit map
  - 16 different pictures is reasonable start
  - Pick the closest one to the current orientation
- Calculating from scratch usually too slow
- Sometimes supported by hardware



Sprite Animation

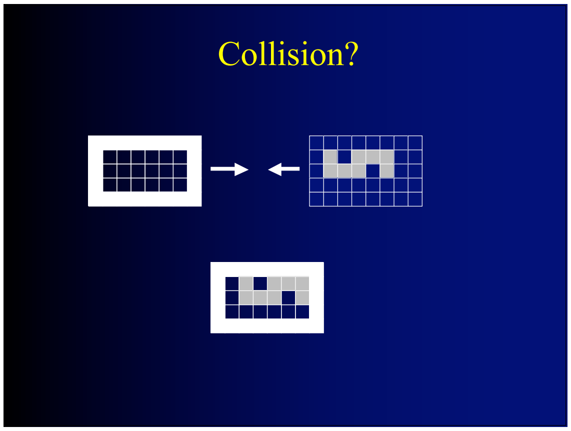
- Changes in the display as state of object changes
  - Example: standing, sitting, jumping, singing, shooting
- Choose the current bit-map based on object state
  - Might require separate timer for animation changes
- Storage if including rotation
  - #_of_bitmaps = #_of_angles * #_of_states



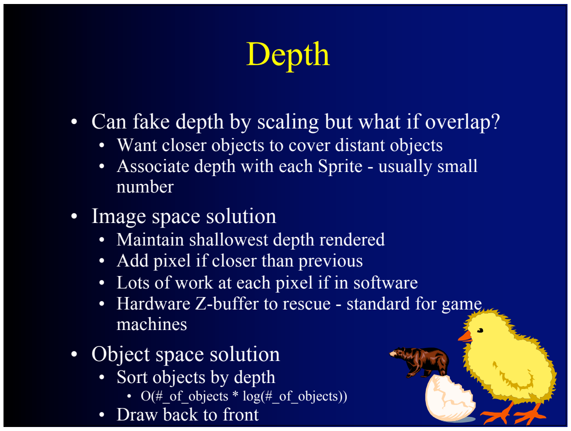
Sprite Collisions

- Easiest:
  - Use the bounding box that includes all the pixels

    Test if vertex of one in bounding box of other
- Tricky:
  - Use something a little smaller to avoid some fake collisions
  - If things happen fast enough, people can't tell
- Almost right but expensive:
  - Test if non-transparent pixels overlap
  - Can still miss some cases...

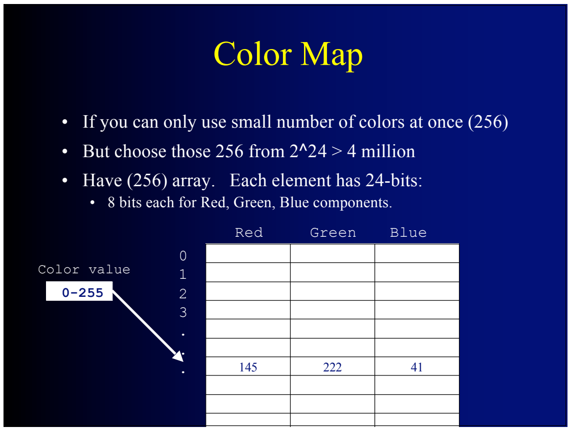


Collision?



Depth

- Can fake depth by scaling but what if overlap?
  - Want closer objects to cover distant objects
  - Associate depth with each Sprite - usually small number
- Image space solution
  - Maintain shallowest depth rendered
  - Add pixel if closer than previous
  - Lots of work at each pixel if in software
  - Hardware Z-buffer to rescue - standard for game machines
- Object space solution
  - Sort objects by depth
    - O(#_of_objects * log(#_of_objects))
  - Draw back to front



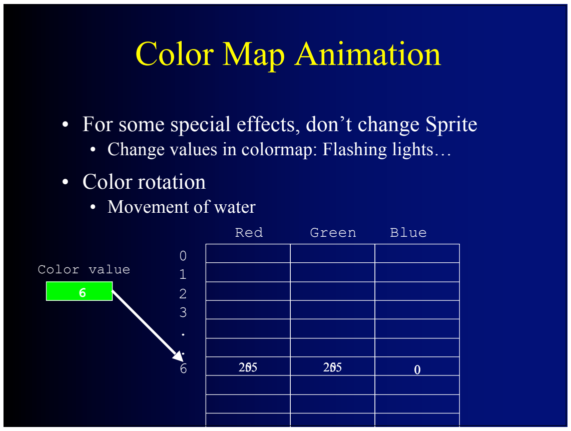
Color Map

- If you can only use small number of colors at once (256)
- But choose those 256 from 2^24 > 4 million
- Have (256) array.   Each element has 24-bits:
  - 8 bits each for Red, Green, Blue components.

Color value

0-255

	Red	Green	Blue
0			
1			
2			
3			
.			
.			
.	145	222	41



Color Map Animation

- For some special effects, don't change Sprite
  - Change values in colormap: Flashing lights…
- Color rotation
  - Movement of water

Color value

6

	Red	Green	Blue
0			
1			
2			
3			
.			
.			
6	255	255	0