Lecture 1 | RTIS

CSD1130
Game
Implementation
Techniques

1. RTIS

1.1. Introduction

- A video game is a real-time interactive concurrent events application simulating a starship fighting enemies in space

1.2. Why *concurrent?*

- The starship, bullets, enemies, sound effect, music, and text exist and move at the same time as a series of coincident events
- The list of coincident events:
    - Testing the keyboard for up, down, left, and right key strokes
    - Scrolling the background
    - New ship position calculation
    - Testing if the fire key is hit
    - Creating a bullet
    - Calculating the new bullet position
    - Testing for collision between each bullet and each enemy
    - Calculating the new enemy position
    - Playing a sound effect when an enemy collides with a bullet
    - Playing the explosion animation when an enemy intersects with a bullet
    - Updating the texts
    - Playing the music
    - Etc..

1.3. Why *interactive?*

- The player decides when and where to move the ship
- The player decides when and where to fire a bullet
- The game AI decides when to attack
- The number of bullets increases each time a bullet is fired
- The score updates each time an enemy is hit
- The enemy explodes when hit by a bullet
- Etc...

1.4. Why *real-time?*

- When the ship moves, its new position is calculated at run time
- The collision between the bullet and the enemy is detected at run time
- The enemy's new position is changed at run time
- The text is updated at run time
- Etc...

1.5. How do we write an application with concurrent events?

- We need to be able to execute several instructions at the same time
- It would be nice if we have a CPU dedicated for each event
- Usually, we only have one CPU (or a dual/quad core CPU)
- But we need to execute several instructions in parallel
- Let's divide the second into 60 pieces
- Each piece would be 1/60 or 0.016 a second (16.66 milliseconds)
- If during each 16.66 ms we update sequentially all game components, the game components would be updated 60 times a second
- The player will get the illusion that the events are coincident or parallel – which means that the events are happening at the same time
- In reality the events are not parallel; they are pseudo-parallel
- The events do not happen at the same time, they happen sequentially
- Because the events are updated at a fixed interval of 60 times a second, the illusion of concurrent events is achieved
- Each 16.66 ms duration is one game iteration
- Since the iteration repeats as long as the game is running, the concurrent events are controlled by repeating the game iteration through a game loop
- The game loop also makes the series of events update as a motion picture or pictures in motion

1.6. Game loop?

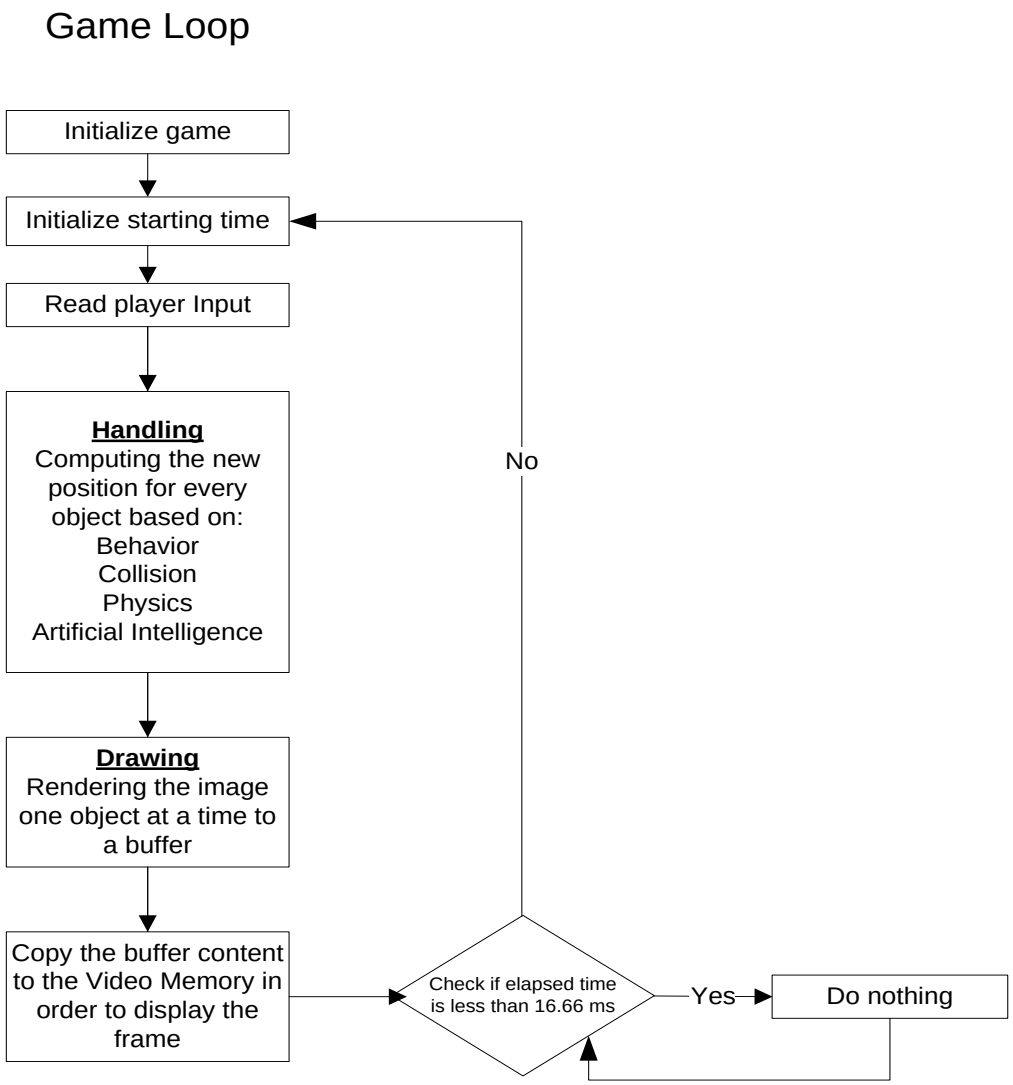
- The game loop iteration duration greatly affects the illusion of concurrent events
- If the duration of the game iteration is long, let's say 0.1 s, then the simulation will feel slow
- Why? Because the reaction to the events happens only 10 times in one second
- On the other hand if the duration of the game iteration is short, like 0.016 s, then the simulation feels smooth
- Why? Because the reaction to the events happens 60 times in one second
- Consequently, the duration of the game iteration is called the *frame time*
- Therefore, the game speed is measured by frames per second
- For example, when we say a game speed is 60 frames per second or 60 f/s or 60 fps, then the game iteration duration is 16.66 ms

1.7. How do we add interaction at real time with concurrent events?

- During the game iteration, we:
  - Detect and register the user input
  - Execute the behavior of each object; usually the object behavior depends on:
    - Input from the keyboard
    - Input from other objects
    - Physics
    - Collision status
    - AI
    - Etc...
  - Once all the objects are updated, their position and status at the current game loop is determined
  - Render the objects
- This means that the objects are rendered as many times a second as the game speed
- For example in a 60 fps game, the objects are rendered 60 times a second
- During the beginning of the rendering during the game loop, a blank frame is prepared
- Then all the objects are rendered sequentially
- When an object moves, its position at game loop *n* is slightly different than its previous position at game loop *n*-1; also, its position at game loop *n*+1 would be different than the position at game loop *n*

1.8. Game flow overview

- Overview of the game flow:

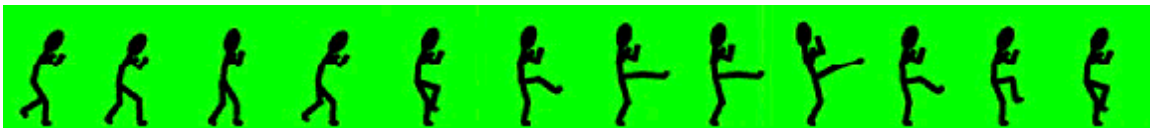


1.9. Motion Pictures

We all go to the movies, but did you know that they were once called “motion pictures”? When you go to a movie, what you actually see is a linear sequence of static images. What does that mean? “Linear sequence” means in a straight line, and “static images” refers to images that are not moving. A motion picture, therefore, is actually a set of non-moving pictures; however, your brain is tricked into thinking that they are moving. This is because the images are flashed so fast that your eyes interpret them as movement, or motion. Additionally, each image is slightly different than the previous one. In other words, image number *n* is slightly different than image number *n*-1.

Motion pictures move at a constant speed of 24 frames per second. This means that during the course of one second, 24 static pictures flash, in sequence, in front of your eyes so your brain interprets it as movement. That means it takes almost 1.5 feet of film for each second of a movie that you watch. If you ever get a chance, take note of how large the reels of film in a movie theater are.

The stick figure below, for example, demonstrates how a kick would look as a few static frames:



The above image consists of 12 frames. This animation would be about half a second of a movie since a movie displays 24 frames per second. Even so, your brain does not see the individual pictures; instead they move so fast that your eyes "paint" a moving image on your brain that appears as movement. The same rules apply in video games. We use single frames to act as static images of a character or object in a game and connect a sequence of them together to create motion. We refer to the motion in games as animation.

1.10. Motion Pictures vs. Games

Games are also made of sequentially displayed pictures. However, the images used to create the illusion of motion in a game cannot be determined in advance. Why not? The reason is that each player plays the game differently. For example, one player might start walking toward the left in a specific level, while another might start walking toward the right. Consequently, each time somebody plays the game, different scenarios will happen. Using this fact, we can conclude that the pictures used to create the motion effect in a game have to be constructed while playing the game in order to reflect the user's interaction and perspective.

Movies display 24 pictures per second, while games usually display 60 pictures per second. This means that we have to construct and display 60 pictures per second. Creating the images, flashing them on the screen as needed, and presenting a separate experience to each player is called creating a **real-time interactive simulation**.

- Each time a game loop is executed, one frame (image, picture) is generated and drawn on the screen
- Executing 60 game loops per second will consequently generate the needed 60 frames per second
- Displaying each generated frame for a small amount of time (1/60 seconds) will create the illusion of motion

- Example:
  - To show a ball moving over a background, we first calculate the new position of the ball and then display the ball at its new position. The calculation and displaying are made in one game loop. By repeating the game loop 60 times per second, the ball appears to be moving.

    At game loop *n*, we have a ball on a background positioned at *P(100,300*). The handling of the ball consists of calculating the new position based on a vector *V*(3,0). Vector *V* means that the ball's position is incremented by 3 along the *x*-axis and incremented by 0 along the *y*-axis.

    - At game loop *n* + 1, the ball's position is *P*(103,300).
    - At game loop *n* + 2, the ball's position is *P*(106,300).
    - At game loop *n* + 3, the ball's position is *P*(109,300).
    - At game loop *n* + 4, the ball's position is *P*(112,300).
    - At game loop *n* + 5, the ball's position is *P*(115,300).
    - And so on.

- At the beginning of each game loop, the display screen is deleted, then the background is displayed, and finally the ball is displayed according to its new position on top of the background. In the example above, notice that the ball's position changes ever so slightly from frame to frame. These steps are repeated at every game loop, 60 times per second.

1.11. Resolution

- What is a pixel?
  - A pixel is the smallest element of a picture.
  - In other words, it is the smallest unit of visual display that can be used to build an image.
  - The word pixel is derived from picture element.
  - Pixels are visible as tiny squares when an image is enlarged.
  - The higher the number of pixels in an image, the better the resolution.

- How is the resolution defined?
  - The number of pixels that the display card can support.
  - As the resolution increases, the quality of the picture increases, as well as the memory used to hold all the information concerning that particular picture.
  - Screen resolution refers to the number of pixels across and down delivered by the display card circuitry.
  - Nowadays, the display card circuitry known as the graphic card can support different resolutions.
  - A typical graphic card can handle resolutions from 640x480 to more than 1600x1200.

1.12. RGB Concept

- How is a color composed?
  - All colors are made of three basic colors: red, green, and blue.
  - The intensity of each of the three basic colors creates another color.
  - Red = 0, Green = 0, and Blue = 0 will produce black.
  - Full red, full green and full blue will produce white.

- Color bit Mode
  - 8 bit (2^8 = 256 colors)
  - 16 bit (2^{16} = 65,536 colors)
  - 24 bit (2^{24} = 16,777,216 colors)
  - 32 bit (2^{32} = 4,294,967,295 colors)

1.13. CRT Concept

- What is the cathode-ray tube?
  - A vacuum tube in which a cathode (electron gun) emits electrons that are accelerated as a beam.

- How does it work?
  - Electrons strike the phosphor coating on the tube.
  - Light is emitted for a short period.
  - The direction of the beam is controlled by two pairs of deflection plates, or by a magnetic field generated by coils
  - The output of the computer is converted by a digital-to-analog converter.
  - The output of the computer is converted to voltages
  - The beam can move along the x and y directions.
  - The intensity of the beam can be turned off.
  - H-blank is the time needed to raster the next row.
  - V-blank is the time needed between the last pixel of the last row and the first pixel of the first row.
  - In order to see a steady image, the same path must be refreshed by the beam at least 50 times a second.
  - Colored CRTs have three different colored phosphors: red, green, and blue.
  - Consequently, there are three electron beams per pixel.

- The difference between interlaced and non-interlaced
    - The refresh cycle of the interlaced display monitor is divided into two frames.
    - All the odd-numbered scan lines are displayed in the first frame, and the remaining lines are displayed in the second frame.
    - When those two frames are drawn in succession, the mind creates an image of the combination of the frames.
    - The non-interlaced display monitor refreshes all horizontal lines sequentially.

1.14. Refresh Rate

- The number of times, per second, that images are refreshed on the display, measured in hertz (Hz).
- 60 Hertz is considered the minimum acceptable level.
- Do not confuse it with the frame rate of the application. They are totally different. The frame rate measures the speed a video source can provide an entire frame for the display.
- Even if the frame itself is not changing, the monitor keeps redrawing it over and over again
- Example:
    - The frame rate of a movie is usually 24, which means it has 24 different frames each second
    - But movie projectors usually have a 48 or 72 Hz refresh rate, which means they project the same frame twice or three times before projecting the next frame

1.15. Refresh Rate & Frame Rate

- As mentioned previously, the refresh rate of the monitor is totally different than the frame rate of an application
- The monitor and the video card do not have to be in sync, which means the refresh rate can be different from the application's frame rate
- Each time the monitor needs to refresh itself, it takes the content of the primary frame buffer (array of colors found on the video card) and displays it

1.16. Out of Sync

- What if the refresh rate of the monitor and the application FPS are out of sync? (And this is mostly the case).
- This means that while the application is producing a certain number of frames per second, the monitor is actually displaying a different number!
- If the application's FPS is different than the monitor's refresh rate, a tearing effect will take place.
- This occurs because when the monitor is refreshing itself, it ends up getting 2 or more overlapped frames.
- If the application's frame rate is way greater than the monitor's refresh rates, some frames will be completely missed, because newer frames got drawn on top of them in the frame buffer (graphics card) before the monitor got the chance to display them.
- Tearing may be somehow acceptable for some games, but it will be highly noticeable and disturbing in fast paced games, because the visual difference between two consecutive frames is relatively high.

1.17. Vertical Sync

- Vertical sync (or simply vSync), the graphics card is told to synchronize itself with the monitor's refresh rate.
- This is done by making the graphics card *wait* for the monitor to signal that it is ready to display a new frame before actually supplying a new one.
- This way, the application can't produce frames more than the monitor can handle, therefore the content of the graphics card's frame buffer will always contain a whole frame, and not several overlapping frames, thus eliminating the tearing effect.
- Assuming the monitor's refresh rate is 60 Hz, any running application with vSync turned on will be capable of producing a maximum of 60 frames per second.
- So far vSync seems like a perfect solution, but what if the application's frame rate is less than the monitor's refresh rate and vSync is turned on?
- When the application generates a new frame, it would have missed the monitor's refresh time by a small amount of time, therefore it doesn't generate the next frame until the monitor displays the current frame.
- In other words, each 2 monitor refreshes will display the same image, which reduces the application's frame rate to half the monitor's refresh rate.
- Example:
    - Application FPS: 50. Frame generation time: 1/50 = 0.02 seconds
    - Monitor refresh rate: 60. Frame display time: 1/60 = 0.016 seconds
    - Let's see the state of the primary frame buffer (PFB) each time the monitor refreshes itself.

Time	Primary frame buffer	Monitor
0.0	Blank	Nothing yet
0.016 (Monitor refresh time)	Blank	Blank
0.02	Contains frame 1	Blank
0.032 (Monitor refresh time)	Contains frame 1	Displays frame 1 (Now the game can start working on frame 2)
0.048 (Monitor refresh time)	Contains frame 1	Displays frame 1
0.052	Contains frame 2	Still displaying frame 1
0.064 (Monitor refresh time)	Contains frame 2	Displays frame 2 (Now the game can start working on frame 3)
0.08 (Monitor refresh time)	Contains frame 2	Displays frame 2
0.084	Contains frame 3	Still displaying frame 2
0.096 (Monitor refresh time)	Contains frame 3	Displays frame 3 (Now the game can start working on frame 4)

    - As you can see, the monitor refreshes itself twice using each frame, which ultimately drops the application's frame rate to 30 (Monitor refresh rate / 2)

- Following the same logic, if the application's frame rate is already lower than 30, it will be forced to drop to 20 (Monitor refresh rate / 3), and so on.

1.18. LCD Monitors

- Unlike CRT monitors, LCD monitors do not have an electron gun
- It contains an array of liquid crystals, between two layers of polarized glass
- Light is sent from the back of the first glass layer passing through the crystals
- On the other hand, the crystals' orientations are modified in order to alter the light that passes through them to the screen
- Therefore, an LCD screen doesn't need to refresh the entire display all the time, it can only modify a part of it by changing the shape of the correspondent crystals
- The time needed by the LCD monitor to change the shape of the crystals is called “Response time”. The response time (in milliseconds) is determined by calculating the amount of time needed by the LCD to go from full black to full white
- Logically, the lower the response time, the better. A lower response time means that the LCD monitor can react faster to color changes coming from the application
- For compatibility reasons with games and hardware, a refresh rate can be set for LCD monitors, especially for games where vSync is enabled
- Remember that when vSync is enabled, the game FPS is locked to the monitor's refresh rate. This is why LCD monitors need to emulate the CRT's refresh rate, although they don't have this limitation