## Graph Theory Applications in Video Games

Clara Nguyễn COSC 594 – 2020/03/11

## Questions

- Given a 3D model *M* of *n* vertices, how many triangles are drawn if done via Triangle List?
- What is the  $lg^*(2^{2^{65536}})$ ? Alternatively, what is the  $lg^*(^62)$ ?
- What does bitDP stand for?



## About me

- Master's Student on Course-Only track.
- Did undergrad at UTK. Graduated in Spring 2018.
- Hobbies
  - Game/Web Development
  - Content Creation (Music & Video)
- Born in Knoxville, TN! Look outside a window for a picture if you want.



#### More on me!

- Been Programming since I was 6. I like to do side projects on the side.
- Not actually a gamer.
- Been a TA here for around 4 years.
- Outside of Computer Science, my goal is to become a polyglot of Asian Languages.
- Not a pet person... (But I prefer cats btw)



#### Showcase - Game Development History

- Involved since mid-2008
- Worked with other Indie teams
- In-house Engine Development.
- 2014 Solo Project: "Keyboard Hero"
  - Rhythm Game like Guitar Hero
  - Released on Gamejolt
  - Coded in GML, Delphi, and C++
  - Over 63,000 views and 16,000 plays



## Showcase - Keyboard Hero V7.5





## Showcase - Game Development History

- 2017 Solo Project: "Project RX"
  - Successor to previous game.
  - Had composers create music specifically for the game.
  - Over 20 songs charted.
  - Engine written in C++ entirely from scratch.
  - Unreleased as of 2020.



#### Showcase - Game Development History

- 2017 CN\_GL (Clara Nguyễn's WebGL Wrapper)
  - Concept 3D engine written entirely from scratch to be playable in your web browser.
  - Written entirely from scratch in 52 hours.
  - This is playable!

http://web.eecs.utk.edu/~ssmit285/vORIcEmA/finalp/



## Why game dev experience matters

- It's one thing to play games. It's another to develop them.
- Code can't be written sloppily. Usually has to generate and draw 30 -60 frames onto your monitor on modern hardware.
  - It's *extremely* obvious when a game is poorly optimised.
- There's lots of unique problem solving in Game Dev. You often build a "toolbox" of ways to approach a problem over time.
- Relevance-wise, Graph Theory plays a huge role in game development.

## Disclaimers

- This is not your average talk.
- This is a Graph "Theory" talk... I only give a handful of game mentions and stick to concepts.
- Algorithm discussion is minimal. If I mention an algorithm, then I'll tell you what it should do, not go over the procedure (except for DFS).
- Topics are laid out intentionally to where they all may not be discussed.
  - All topics and details are here: <a href="https://tiny.utk.edu/talk5">https://tiny.utk.edu/talk5</a>

## Outline

- *The Warmup* "Rules" of Graphics
- Racing Games Lap Counting
- Maze Generation Disjoint Sets & Union-Find
- Hamiltonian Path Detection bitDP
- Honourable Mentions
- Discussion

# "Rules" of Graphics

## The "rules" of graphs of computer graphics

- Unlike most graphs we dealt with in class, the rules change here:
  - Vertices have **positional** coordinates (x, y, z) to define position in space.
  - There is only **one** way to represent graphs in space.
  - Edges (connections between vertices) are **implied**.
  - Everything is oriented around **triangles**.

### The "rules": Edge Implication

- Edge Implying depends on how we tell the computer to draw.
- Several modes. Here are the common ones:
  - **Triangle List:** Every 3 vertices form a triangle.
  - **Triangle Strip:** First 3 vertices form a triangle. Every new vertex after will form a triangle with the previous 2 vertices.
  - **Triangle Fan:** First vertex is in every triangle. Each set of 2 vertices after the first form a triangle with the first vertex.

## The "rules": Triangle List

- Naïve triangle drawing in multiples of 3.
- n/3 triangles drawn.
- Assume we are given a model *m* where  $V(m) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$



## The "rules": Triangle Strip

- Uses previous 2 vertices & new one to form triangles.
- $n \ge 3$ . n 2 triangles drawn.
- Assume we are given a model m where  $V(m) = \{v_1, v_2, v_3, v_4\}$



#### The "rules": Triangle Fan

 $V_3$ 

- Uses first vertex and latest 2 vertices to form triangles.
- $n \ge 3$ . n 2 triangles drawn.
- Assume we are given a model m where  $V(m) = \{v_1, v_2, v_3, v_4, v_5\}$

 $V_3$ 

 $V_{4}$ 

 $\mathbf{V}$ 

 $V_3$ 

 $V_{A}$ 

## The "rules": Coordinate System

- Two of the most popular cartesian coordinate systems for 3D space:
  - (x, y, z) where  $\boldsymbol{z}$  is the height axis
  - (x, y, z) where  $\boldsymbol{y}$  is the height axis

 $\mathbf{Z}$ 





### The "rules": Back-Face Culling

- Front side has a triangle. Back side is invisible due to **back-face culling**.
- Relies on the order we draw the vertices. Vertices with order being
  - clockwise is front-facing. Counter-clockwise is the back.



# Lap Counting

## **Racing Games**

- A racing game must keep track of a few things...
  - Player Lap
  - Player Position
  - Distance between players
- How do games know when a player has completed a lap?

• Assume the following (extremely simple) racetrack:



- Divide the track into "checkpoints".
- Players will have to hit all "checkpoints" and the finish line for a lap to count.
- This can be implemented as a directed graph where all checkpoints are vertices and a complete lap is a **Hamiltonian Circuit**.

• Simple racetrack broken up into checkpoints, and as a directed graph:



#### Distance between players

- How will we know how far someone is from first place?
- Graph is broken segments by-vertex, rearranged into a straight line with circular ending node, making distance computation extremely trivial.



## Distance between players

• So let's say Dr. Langston ( $P_L$ ) having a really good race... unlike me ( $P_N$ )...



## Lap Counting – Breaking the Rules

- In practice, there are other ways to do lap counting... besides Hamiltonian circuit detection.
- They flopped. Let's look at an extreme example.

## Mario Kart Wii for Nintendo Wii (2008)



#### Mario Kart Wii – Breaking it down

- Breaks track into **spawn checkpoints**.
  - If you fall out of the track, you spawn at these.
- Breaks track into key checkpoints.
  - Finish line also counts as a key checkpoint.
  - Tells where you are and if you completed the lap... or do they?

## Mario Kart Wii – Breaking it down

Assume the following (extremely simple) racetrack with key checkpoints,

spawn checkpoints, and a finish line:



### Mario Kart Wii – Breaking it down

 Going between a key checkpoint and a next checkpoint (spawn, key, finish) updates where you are in the track.

 Example: Hitting between 1 and the spawn checkpoint right after will register as you passing through checkpoint 1.



#### Mario Kart Wii – Ultra-Shortcuts

 Critical Flaw: Game allows you to hit the next, current, and previous key checkpoints. Completing a lap requires hitting only the last one.

 From the start of the race, we can avoid going through 1 and 2. Just jump to 3 and drive up to 0. The lap will count.

• This is known as an Ultra-Shortcut.



#### Mario Kart Wii – Ultra-Shortcuts

• These are not as simple as driving backwards though.

 Going in reverse from the finish line will deduct 1 from your lap count. Detected by the spawn checkpoint right behind.

• Usually involves finding glitches or out-of-bounds areas to jump to 3.



#### Mario Kart Wii – Ultra-Shortcuts

• The *normal* world records didn't last very long after that...

2008-06-01	1′35″799	Ridley	$\rightarrow$	2019-09-13	0′17″100	Ejay
2008-06-01	1′29″550	Ridley		2019-09-19	0′16″852	Ejay
2008-06-01	1′26″078	Alvin		2019-09-23	0′16″691	Niyake
2008-06-01	1′03″520	Ridley		2019-09-23	0′16″591	Niyake
2008-06-01	0′43″912	Alvin		2019-09-23	0′16″385	Niyake
2008-06-01	0′42″446	Ostro		2020-01-12	0′16″332	Niyake

• Moral of the Story: Use Hamiltonian Circuit detection for lap counting.

## Maze Generation

### Disjoint Sets & Union-Find
### An observation of mazes

- Cells matched with a select few of adjacent cells.
- Others are separated by "walls".
- Can be represented as a graph. Depending on properties of the maze, it can be a minimum spanning tree.
- We can use DFS (Depth-First Search) and BFS (Breadth-First Search) to traverse the maze to find a solution easily from any *S* to any *T*.

## **Disjoint-Sets**

- Sets that have no element in common.
- "Mazes" with every wall put up is a good example, as no cell is connected.
  - Basically a graph without any edges connecting any nodes.
- We have operations: **union** and **find**:
  - Union: Join two disjoint sets together.
  - Find: Get the ID of the set that a cell belongs to.

### Disjoint-Sets – Continued

- Union: Join two disjoint sets together.
  - Notated as union(i, j) where  $S_i = S_i \bigcup S_j$ .
  - In English: All vertices in  $S_j$  move into  $S_i$ . Then,  $S_j$  is *deleted*.
- Find: Get the ID of the set that a cell belongs to.
  - Notated as find(i) where i is a cell ID.
  - More on this in a bit...

• Assume a graph M where n = 16, and m = 0. Each separate vertex is part of

its own set  $S_i(v_0 \in S_0, v_1 \in S_1, ..., v_{n-1} \in S_{n-1})$ . Show as a  $4 \times 4$  grid:

0	<sup>1</sup> 1	<sup>2</sup> 2	<sup>3</sup> 3
<sup>4</sup> 4	<b>5</b>	<sup>6</sup> 6	7 7
8	<sup>9</sup> 9	<sup>10</sup> 10	<sup>11</sup> 11
<sup>12</sup> 12	<sup>13</sup> 13	<sup>14</sup> 14	1515

• Let's do union(1,2). Notice how the walls break down between the two.

They have an edge between them. Now  $S_1 = \{1, 2\}$  and  $S_2$  is deleted.





• Let's do union(2, 6). Break down the wall between where 2 used to be and 6.

Now  $S_1 = \{1, 2, 6\}$  and  $S_6$  is deleted.





- To properly generate a maze:
  - Repeat the procedure on cells that are

adjacent but are in different groups.

• Do this until there is only one group

left...  $S_0 = \{v_0, v_1, ..., v_{n-1}\}$ 



## Disjoint-Sets – Some properties

- Known as Randomised Kruskal's algorithm.
- There are no cycles.
- There is one path from every S to every T.
- Tends to generate mazes with patterns that are easy to solve.
- If shown as a graph, it's a minimal spanning tree.



### Disjoint-Sets – We can do better

- A simple maze is boring.
- We can connect 2 together by breaking down a wall between them (or even adding a "hall" between them).
- Any cell in one maze is always accessible from any other cell. Connecting like this keeps this property intact as we can always go toward the "hall".
- This makes more complex, interesting, non-square puzzles.

## Disjoint-Sets – 2D Expansion

• Horizontal Expansion. Notice how there is always a path from the left maze to the right maze since we can always access 7 and, thus, the "hall" to 20.



### Disjoint-Sets – We can still do better

- We can expand a dimension (or a few).
- Connect 2 mazes together by making a cell have an "elevator" to go up.
- Same property from before still holds. There will always exist a path from one cell to another, even when going up to another floor.

## **Disjoint-Sets – 3D Expansion**

• Floor Expansion. Again, notice how there is always a path from every cell to every other cell.



- In theory, union(i, j) on two sets is trivial. To a computer, it requires work.
- Find: Get the ID of the set that a cell belongs to.
  - Notated as find(i) where i is a cell ID.
  - Interpret the set as a graph.
  - Go up to root of the "graph". That is the set's ID.
  - When doing a union(i, j), the ID of two node's set IDs must be different or else a cycle will occur. The lowest index (rank) becomes the new root.

• Interpret maze M as a traditional graph with vertices and edges.



• Let's do union(1,2). Then, find(2) = 1 as  $v_2 \in S_1$ .



• Okay, now do union(2,6). Then, find(6) = 1 as  $v_6 \in S_1$ .



• Keep building the minimum spanning tree until entire graph is connected. For every vertex in the final graph, find = 0 as they are all in  $S_0$ .



• This can become bad quickly... The vertex at the bottom right of the maze has to traverse through **6** vertices to reach the root.



- As usual, we can do better... *much better*.
- Let's apply two concepts: Union by rank and Path compression.
  - Union by rank Attach shorter tree to the root of the taller tree.
  - **Path compression** Make every node point straight to the root.

- The original lookup speed requires around *n* lookups to reach the root.
- With our optimisations in place, it becomes  $\lg^* n$  (iterated logarithm base 2).
- In the world of Computer Science, this is essentially **constant time**.

$^{\mathrm{n}}\mathrm{a}$	Х	lg* x
$^{1}2$	2	1
$^{2}2$	4	2
$^{3}2$	16	3
$^42$	65536	4
${}^{5}2$	$2^{65536}$	5
${}^{6}2$	$2^{2^{65536}}$	6

# bitDP

## Hamiltonian Path Detection

Some of you may have seen this before...

### Hamiltonian Paths

- A path where we visit every vertex once.
- NP-Complete.
- For computers, naïvely finding these in a graph of size *N* **explodes** into *N*! steps.
- Detection useful for a game generating random paths and needs to check for correctness before giving to the player.



### Naïve Brute-Force Method

- Perform a DFS (Depth-First Search) from the starting vertex *S* search around all possible combinations of paths until we find a Hamiltonian Path.
- Gets the job done, but is nowhere near efficient.

### DFS Breakdown

- Assuming a graph G, keep a list  $V'(G) = \{\}$  which is the path (in the order we visited the vertices). Mark all vertices as **unvisited**.
- Behold the procedure DFS(v). Run it on DFS(S):
  - 1. Mark v as **visited** and add it to the end of V'(G).
  - 2. Go through every **unvisited** vertex v' that v is connected to and do DFS(v').
  - 3. If the size of V'(G) is equal to the number of vertices in G, a Hamiltonian Path exists!
  - 4. If one wasn't found, remove v from V'(G), mark it as **unvisited**, and go back to the previous call of the procedure.

### DFS Example - Setup

• Behold a graph G where  $S = v_0$  and  $V'(G) = \{\}$ . Find if a Hamiltonian Path exists starting from S via DFS(S).

# DFS Example - DFS(S)

3

- $v = v_0$
- $V'(G) = \{v_0\}$
- Call  $DFS(v_1)$

# DFS Example - $DFS(v_1)$

3

- $v = v_1$
- $V'(G) = \{v_0, v_1\}$
- Call  $DFS(v_2)$

# DFS Example - $DFS(v_2)$

- $v = v_2$
- $V'(G) = \{v_0, v_1, v_2\}$
- Call  $DFS(v_3)$

# DFS Example - $DFS(v_3)$

- $v = v_3$
- $V'(G) = \{v_0, v_1, v_2, v_3\}$
- The size of V'(G) is 4. Hamiltonian Path found.



# DFS – Performance Analysis



### Let's bash DFS for a sec

- Multiple repeated function calls
- We have to check if we visited a vertex or not
- This is naïve brute-force. We aren't taking advantage of any "properties".
- We can do better... *much better*.

## Dynamic Programming (DP)

- Mathematical Optimisation by Richard Bellman
- Break a problem down into easier "sub-problems", solve those, and use the result to solve the original problem.
- "Sub-problems" are broken down into even easier "sub-problems" if possible, recursively.

### Held-Karp Algorithm

- Proposed by Michael Held and Richard Karp, as well as independently by Richard Bellman in 1962.
- Utilises DP to solve "sub-problems" of a graph, preventing repeating traversals if a solution is already known.
- Reduces DFS's O(N!) time to  $O(2^N \times N^2)$ . A significant improvement.
- This was mainly for solving TSP (Travelling Salesman Problem). But the variant here will solve for Hamiltonian Paths.

## Held-Karp – An Observation

- **Observation:** Assume a graph G, a subgraph G', and H = G G'.
- If there is a Hamiltonian Path in G' and a vertex in G' is adjacent to a vertex v in H in G, then there is a Hamiltonian Path in a subgraph G' + v.



## Held-Karp – Example

- Assume a graph G, a subgraph G', and H shown below.
- It's trivial to tell that G' has a Hamiltonian Path  $\{v_0, v_1\}$ .



### Held-Karp – Example

- Now let's look at a new sub-graph, I where  $V(I) = \{v_0, v_1, v_2\}$ .
- We know there was a Hamiltonian Path in G'. I has the same vertices plus  $v_2$ .

Since any vertex in  $G'(v_0 \text{ or } v_1)$  can reach  $v_2$ , it also has a Hamiltonian Path.


### bitDP

- B I T どうてきけいかくほう
  bitDP = Bit Dynamic Programming (ビット動的計画法)
- Use a DP table where **vertices** go on one side and **bitmasks** go on the other.
  - Bitmask represents subgraphs of G.
- Table is sized  $N \times 2^N$ .
  - e.g. Graph with 4 vertices has 16 subgraphs, from 0000 to 1111.
- At the final mask (1111), if **any** value is set to 1, there is a Hamiltonian Path in the graph *G*!

## bitDP – Example (Reading the table)

• Make a bitDP table based on the graph:

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	А	В	С	D	Е	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3

0

2

## bitDP – Example (Reading the table)

• Make a bitDP table based on the graph:

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Vertex/Mask	0	1	2	3	4	5	6	7	8	9	А	В	С	D	Е	F
0	0	1	0	1	0	1	0	1	0	0	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	1	1
2	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1	1
3	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1

3

0

2

## bitDP – Example (Reading the table)

- Consider Mask at 0xB (1011):
  - Vertices Visited: 0, 1, 3

9  A  B  C    0  0  1  0    0  1  0  0	
0 0 1 0 0 1 0 0	D
0 1 0 0	1
	0
0 0 0 1	0
0 1 1 1	1



3

2

- Is there a path between those three that:
  - Ends at 0? Yes
  - Ends at 1? No
  - Ends at 3? Yes

## Held-Karp (via bitDP) – Performance Analysis



## **Honourable Mention**

## Maze Generation, Part II

You thought I was done...

## Entombed for Atari 2600 (1982)



#### Entombed for Atari 2600

- Released in 1982.
- Simple design. Player moves through a maze trying to avoid enemies. Contact with enemies results in a game over.
- Maze moves upwards.
- If a player is stuck in a dead end, it's also a game over.



#### Entombed for Atari 2600 – The Technical Details

- Storing all possible mazes in memory is impossible.
- Mazes were generated "on -the-fly".
- Right side is just a mirrored version of the left side.
- Didn't use Disjoint-Sets with Union-Find. How did they do it?



#### Entombed for Atari 2600 – Maze Generation

- Programmer was **drunk** and developed an "algorithm" for it.
- A cell is set by looking at 5 nearby squares, then looking up information in a lookup table.
- Generates a playable maze... every time... somehow.



#### Entombed for Atari 2600 – Maze Generation

- Why does this work? No one knows why.
- When programmer was interviewed, he said it came from another programmer.
- Said "He told me it came upon him when he was drunk and whacked out of his brain".
- It's even on the Wikipedia page for "List of unsolved problems in computer science".



## Entombed for Atari 2600 – Lookup Table

		с	d	е	a
1	a	b	x		0

	_	a	b	с	d	е	х	 a	b	с	d	е	х
1		0	0	0	0	0	1	1	0	0	0	0	1
		0	0	0	0	1	1	1	0	0	0	1	1
		0	0	0	1	0	1	1	0	0	1	0	1
		0	0	0	1	1	?	1	0	0	1	1	?
		0	0	1	0	0	0	1	0	1	0	0	0
		0	0	1	0	1	0	1	0	1	0	1	0
		0	0	1	1	0	?	1	0	1	1	0	0
		0	0	1	1	1	?	1	0	1	1	1	0
		0	1	0	0	0	1	1	1	0	0	0	?
		0	1	0	0	1	1	1	1	0	0	1	0
		0	1	0	1	0	1	1	1	0	1	0	1
		0	1	0	1	1	1	1	1	0	1	1	?
		0	1	1	0	0	?	1	1	1	0	0	?
		0	1	1	0	1	0	1	1	1	0	1	0
		0	1	1	1	0	0	1	1	1	1	0	0
		0	1	1	1	1	0	1	1	1	1	1	0

#### How does it relate to Graph Theory?

- It's unsolved, and we know other maze generation algorithms are constructed from graphs, maybe there's an explanation that involves Graph Theory?
- Apparently, you have to be drunk to make cool stuff...

# One more thing...

## Leaked Exam

March 11, 2020	55 Name Clara Nguyễn
Compute Maze Generation. Define M M will always be a playable	er Science 594/690 - Exam 1 as a maze generated with the Entombed truth table. Prove that maze from top to bottom. I have no idea. Have mary please.

#### References

- "Held-Karp Algorithm." *Wikipedia*, Wikimedia Foundation, 19 Feb. 2019, <u>https://en.wikipedia.org/wiki/Held%E2%80%93Karp\_algorithm</u>.
- "Mario Kart Wii." *Wikipedia*, Wikimedia Foundation, 11 Feb. 2020, <u>https://en.wikipedia.org/wiki/Mario\_Kart\_Wii</u>.
- Summoning Salt. "Mario Kart Wii: The History of the Ultra Shortcut" YouTube, 11 Feb. 2020, https://www.youtube.com/watch?v=mmJ\_LT8bUj0.
- "Entombed (Atari 2600)." *Wikipedia*, Wikimedia Foundation, 11 Feb. 2020, <u>https://en.wikipedia.org/wiki/Entombed\_(Atari\_2600)</u>.
- Aycock, John and Tara Copplestone. "Entombed: An archaeological examination of an Atari 2600 game." *Programming Journal 3* (2018): 4.
- "Nguyễn, Clara". "Hamiltonian Paths & bitDP." Hamiltonian Paths & bitDP, 11 Feb. 2020, <u>http://utk.claranguyen.me/talks.php?id=bitdp</u>.
- "Grumble Volcano." MKWii WR History, https://mkwrs.com/mkwii/display.php?track=Grumble+Volcano.

# Discussion

## Questions

- Given a 3D model *M* of *n* vertices, how many triangles are drawn if done via Triangle List?
- What is the  $lg^*(2^{2^{65536}})$ ? Alternatively, what is the  $lg^*(^62)$ ?
- What does bitDP stand for?

## Graph Theory Applications in Video Games

Clara Nguyễn COSC 594 – 2020/03/11