Project 5

Sunday, November 27, 2022 11:43 AM

Data Structures

Arrays [ ]

- V = V-table
    - Simple array to store all the vertex IDs, and vertices for 1 triangle are consecutive
    - Ex: [1, 2, 3, 4, 5, 6]
        - Vertex IDs 1, 2, 3 belong to first triangle, and 4, 5, 6 belong to 2nd triangle

- G = Geometry-table
    - Multi-dimensional array to store the actual 3D coordinates of a vertex
    - Ex: [[x1, y1, z1], [x2, y2, z2]]
        - G[0] = the actual coordinates corresponding to vertex 0 [x1, y1, z1]
        - So, if you want the actual coordinates corresponding to a specific corner, use:
            - G[V[c]]

Dictionary { }

- O = Opposite-table
    - For each index, c, in O
    - Key: V[c]
    - Value: the computed value (use helper function to compute O table)

Variables

- currentCorner (integer)
- Flags (boolean)
    - currentCornerVisible
    - showRandomColors

Helper Functions

- See slide 6 in Rossignac's slides

def nextCorner(cornerNum)

- triangleNum = cornerNum // 3
    - Find triangle number based on cornerNum
    - Note: the // is floor division in python
- Return 3 * triangleNum + ((cornerNum + 1) % 3)

def previousCorner(cornerNum)

- Use same idea as nextCorner function
- But, instead of adding 1 to cornerNum in the return statement, you should subtract
- Return 3 * triangleNum + ((cornerNum - 1) % 3)

def oppositeCorner(cornerNum)

- Use the opposite-table dictionary
- Return O[cornerNum]

def swingCorner(cornerNum)

- Return nextCorner(oppositeCorner (nextCorner(cornerNum)))

def computeOTable(G, V)

- Temporary variable to store triplets
- For loop: iterate from 0 to len(V)
  - Append (min(V[nextCorner(i)], V[previousCorner(i)]), max(…), i) to triplets
    - See slide 8
- Sort the triplets (see slide 8)
- For loop: iterate from 0 to len(sortedTriplets), add 2 to iterator each time
  - cornerA = sortedTriplets[i][2]
  - cornerB = sortedTriplets[i+1][2]
  - Assign O[cornerA] = cornerB and vice versa

Modifying the read_mesh function

- In for loop iterating from 0 to num_vertices,
  - Update G table by appending (x, y, z)
- In for loop iterating from 0 to num_faces,
  - Update V table by extending it by (index1, index2, index3)
- Outside of these loops, instantiate the O table by calling the helper function you wrote

Modifying the Draw Function

- Use for loop. Iterate starting from 0 to len(V-table), adding 3 to the iterator each time (eg 0, 3, 6, …)
  - Let iterator variable be c
- beginShape()
- Use if/else block to fill with random colors if showRandomColors flag is turned on
  - fill(random(255), … , …)
  - Otherwise, normal fill is fill(255, 255, 255)
- Use vertex function to draw 3 vertices
  - vertex(G[V[c]].x, G[V[c]].y, G[V[c]].z)
  - vertex(G[V[c + 1]].x, G[V[c + 1]].y, G[V[c + 1]].z)
  - 3rd is similar
- endShape()
- Logic to make the current corner visible if currentCornerVisible is turned on
  - pushMatrix()
  - currentVertex = G[V[currentCorner]]
  - translate(currentVertex.x, …, …)
  - sphere(0.1)
  - popMatrix()

Create a function for subdivision

- Variable for numEdges, which is len(V) // 2
- Make a temporary data structure for the newGTable and newVTable
- Initialize empty dictionary for midpoints

- For loop: going through the O table. Need to have a and b as iterators
  - Endpoint1 = G[V[previousCorner(a)]]
  - Endpoint2 = similar but use nextCorner function
  - Calculate midpoint which is endpoint1 + endpoint2 * 1/2
    - Use Pvector mult

    - Find midpointIndex which is len(newGTable)
    - Append the midpoint to the newGTable
    - Update the midpoints dictionary with the midpointIndex
        - Midpoints[a] = midpointIndex
        - Do the same for midpoints[b]

- For loop: go from 0 to len(V) and add 3 each time to the iterator x
    - Make 2 new index variables to make your life easier
        - y = x + 1
        - z = x + 2
    - Note that newVTable is a list in Python. So you can use the extend function to attach more items to this list
        - We need to add 4 sets of items to newVTable
        - (V[x], midpoints[z], midpoints[y])
        - (midpoints[z], V[y], midpoints[x])
        - (midpoints[y], midpoints[x], V[z])
        - (midpoints[x], midpoints[z], midpoints[y])

- Return newGTable, newVTable, computeOTable(newGTable, newVTable)
    - In the handleKeyPressed section, for key 'd', you can update the global variables you made for G, V, and O by calling this subdivide helper function