# Assignment 01

## Programming:

### Part 01:

Work out how to create a triangular pyramid (this only uses 3 triangles) – you need to define the vertices (you only need 4 vertices for this). Put those in a single buffer. Note that you can always scale them with a matrix later, so you can use whatever scale you want, but generally pick something sane (in coordinates in the range 1-20 ish), and don’t worry about precision to more than 2 decimal places.

### A.

Draw this triangular pyramid on screen, you’ll need to color each of the triangles slightly differently so you can see what it looks like (otherwise it just looks like a triangle).

### Answer:

The output is in the zip file.

### B.

following the controls from the text supplied code, make a little menu that the user can select a direction of rotation (clockwise or anticlockwise) and have it rotate appropriately, as in the lab, pick an angular velocity (angle per time) that’s reasonable to look at. We haven’t had time to talk about how to set things to a clock rather than system speed (and WebGL has some special behavior because it’s in a browser that has its own refresh rules, and browsers mostly don’t run in full-screen mode, that has its own rules…).

### Answer:

The output is in the zip file.

### C.

Either in combination with b) or separately, have your pyramid gradually shrink and expand between -50% in size to + 50% in size (compared to whatever baseline size you started with). You’ll need to apply a scale transformation.

### Answer:

The output is in the zip file.

### Part 02:

Very slightly more advanced: Same thing as part 1 but make a square-based pyramid (so 5 vertices).

### Answer:

The output is in the zip file.

## Theory:

One of the problems we are going to need to think about is the efficient representation of an object and the traversal of the list of vertices that define it. Each vertex coordinate (typically 3 32-bit floats: x,y,z), is connected to some other set of vertices. An object is then defined as a series of vertices, and drawing the object is traversing the list of vertices and their neighbors in the right way.

Broadly there are two categories of algorithms here: matrices and lists.

### 1.

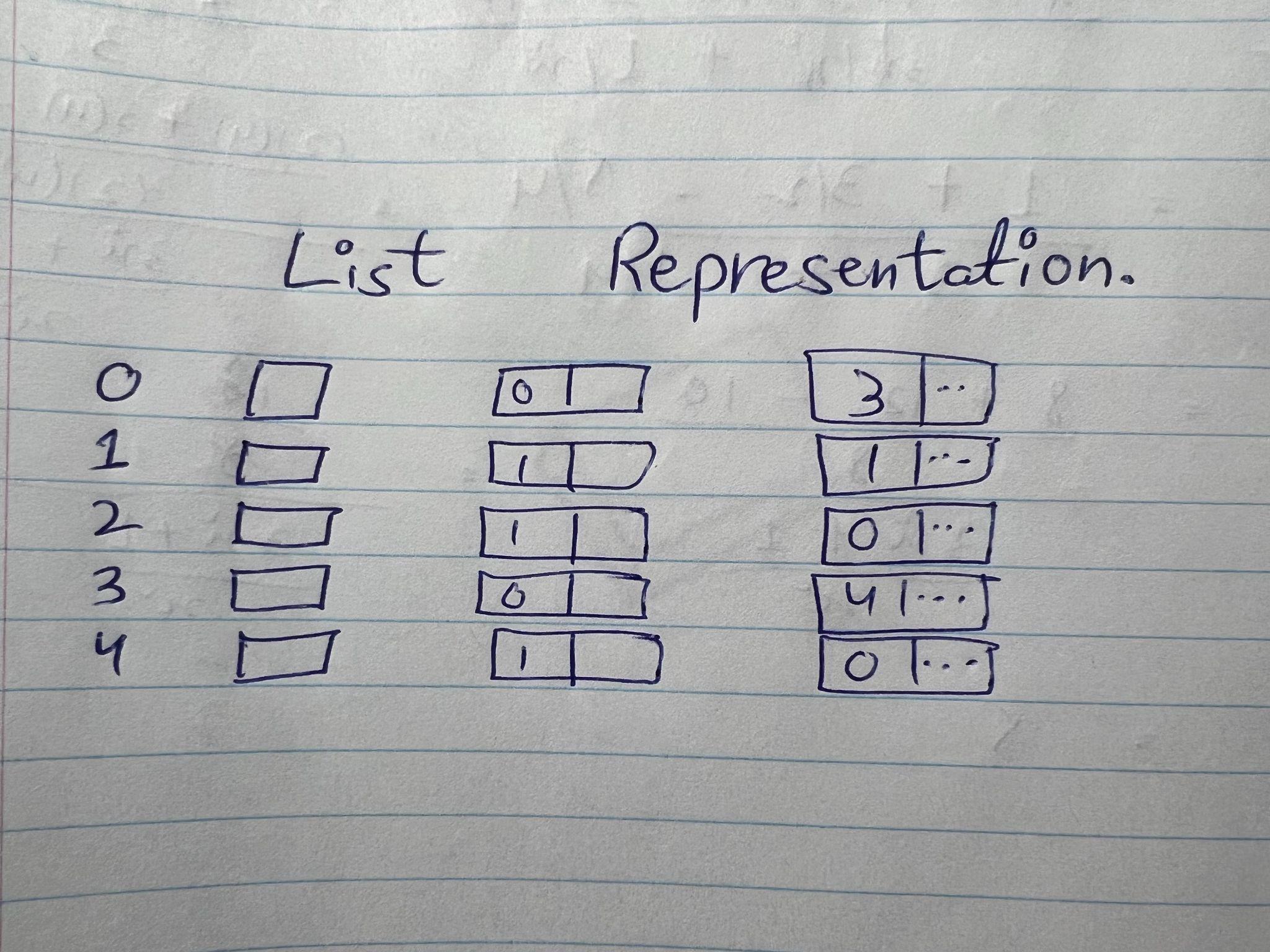
Describe how you could in general use this matrix vs list representations and traverse them to represent an object (those of you who have taken 3020, which should be but isn’t a prereq for this course will recognize this as adjacency lists vs adjacency matrices). 2 marks, I’d say roughly a page to answer, that could include diagrams, so 500 words ish, but less if you’ve got good diagrams explaining it.

### Answer:

We can represent objects on the screen using multiple data structures and algorithms, matrices and lists are those ways to represent something on the screen. Matrices and lists have their own uses and applications to show an object with their own pros and cons. We cannot describe them as a comparison with each other which is better because both have their different usages in different scenarios.

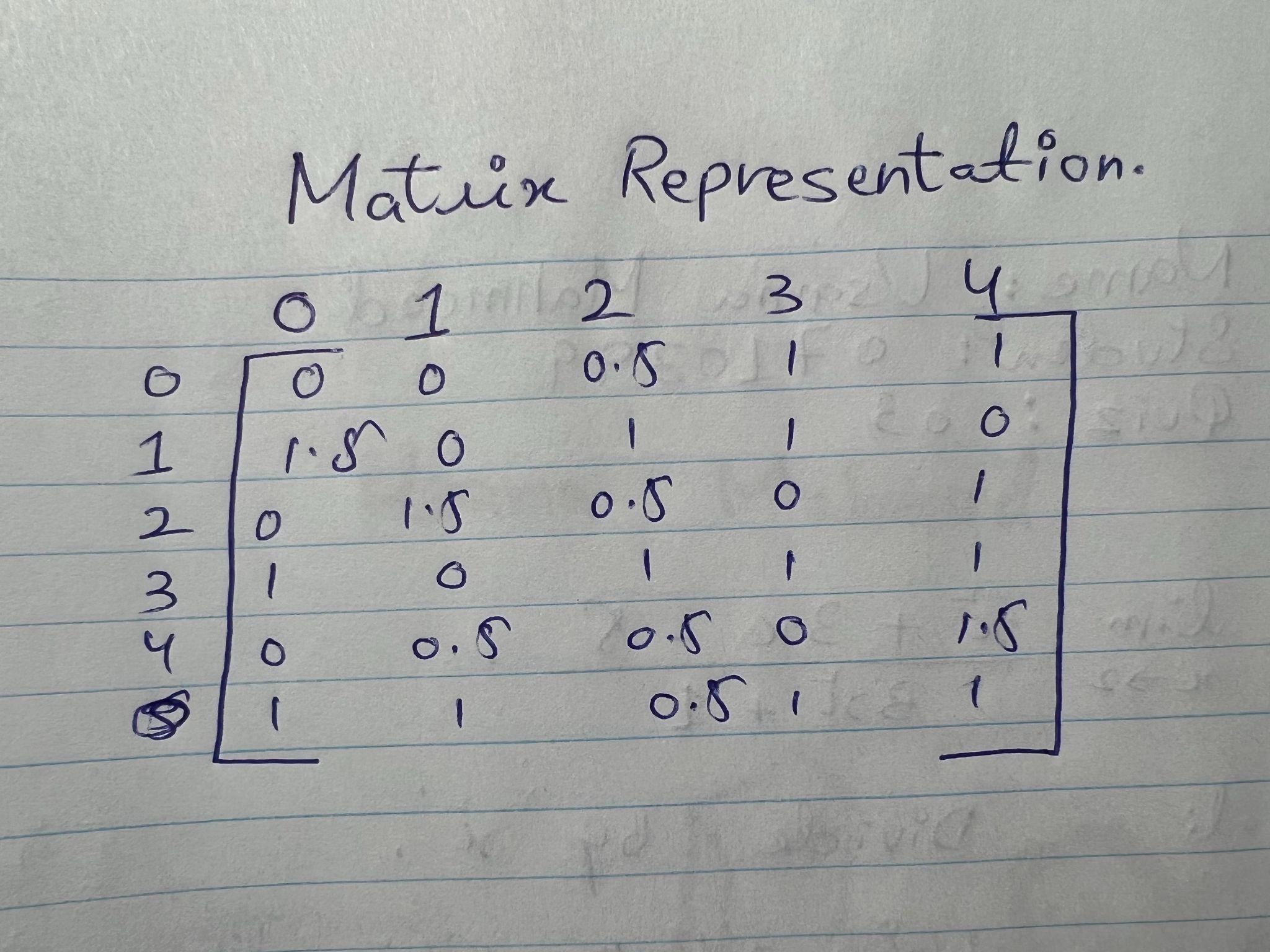
The list is an array, every node of the list contains the vertex of the connected node to it and that connected node contains the vertex of the other connected node, and so on.

We have practiced drawing objects and different shapes on the screen with the help of lists of their vertices and the use of lists of this seems efficient as a representation of a single object on the screen it could be done by lists better than matrices. This is because lists contain more information for a single object as compared to matrices.



Matrices are used to transform the object as well as for many other purposes like data visualization and computation for image construction on the web. There are many types of matrices that can be used like 2D and 3D, we can make as many grids as we want like 4x4 or more. It is basically a 2D array VxV where v is the number of vertices.

Matrix is better when we have to use multiple objects and make connections between them and show relationships between them, in order to represent something, in that particular scenario matrices can do a very wonderful job in connecting different objects with each other.



If we compare both data structures to each other then it can be seen that the complexity of the matrix is more than the lists in worse cases. As if we need to add or remove a vertex in the matrix then the whole matrix needs to be copied. In the case of lists, there are two pointers such as font node and rear node then the addition of a vertex can be done directly. In order to remove a vertex, then we have to find it and then traverse the list for it.

### 2.

How is this a storage vs compute problem? (you can use your programming questions to illustrate different strategies here). (1 mark ~100-200 ish words). Modern objects can have many millions of polygons (e.g. the mako reactor in FFVII remake is 9 million polys), UE5 Nanite and competing technologies allow for fast processing search and culling of hundreds of millions or billions of polygons in real-time, which is why this is important, faster traversal of a list at the expense of some memory might be good, but more memory is well more memory.

### Answer:

When 3D becomes a part of our lives then the need for faster processing speed comes. Then more advanced GPU came into the market which can calculate faster this allows faster graphic processing speed without sacrificing performance and detailing. As modern-day games and other graphical things contain hundreds of millions of polygons which represents the intensive details, no doubt this is a challenge for the computer industries to make efficient GPUs that can calculate billions of polygons. In theory, we can calculate billions of polygons and practically this number is only a few hundred million. The polygons are calculated for the display memory by the GPU so there is a computer problem.

Storing hundreds of millions of polygons rendering objects is also a problem, though GPU can process the object of billion of polygons only if it has enough storage to store it in real-time. We need an efficient data structure to store the data of that billion polygon object, circle tree is a data structure that stores polygons in an efficient way.

### 3.

When we define objects, they are usually defined around the center of the object, or the object is otherwise near the origin, typically we use 32-bit floats for coordinates. When we move that into a world space that might be quite large the space is defined with 32-bit floating point values as well (and the camera is not always at the origin of the world). Explain why floating-point precision and the non-uniform distribution of values potentially pose problems in how we store and draw objects. https://www.researchgate.net/publication/222682924\_Lossless\_compression\_of\_predicted\_floating-point\_geometry might be a useful starting point. You should know from 2300H about floating point formats. 1-mark (100-200 ish words).

### Answer:

Rounding error in floating point is the reason for problems in drawing objects by using floating point precision, it occurs when decimal point wanders when it owing to the big numbers, then this causes the error. The exponent in a floating point representation controls the range, whereas the number of significant bits regulates precision.

Furthermore, when the object is drawn on the screen then due the non-distribution of values causes clumping of the objects, which leads to unclear scenes on the screen objects. The misinterpretation of various values creates other than the actual expected result due to the non-uniform distribution of values on an object that eventually causes the problem in storing and drawing the object. For instance, the image that appeared on the screen has a lot of dark pixels so it might be illuminated as black pixels even in reality if it is meant to have a different color.