## Task 1 – Pipeline

**a)** **[1p]** The real-time graphics **pipeline** consists of three major block. Name them.
Answer: application stage, geometry stage, rasterization stage

**b)** **[1.5p]** Give examples of what is done in each part.
Answer: Application stage – e.g. VFC, animation.
Geometry stage: transformation + per vertex shading (lighting).
Rasterization stage: rasterization, texturing, interpolation of per-vertex values from vertex shader, z-test, fragment shading.

**c)** **[1.5p]** For each part, describe how you can determine if this step is the performance bottle-neck for the rendering.
Answer:

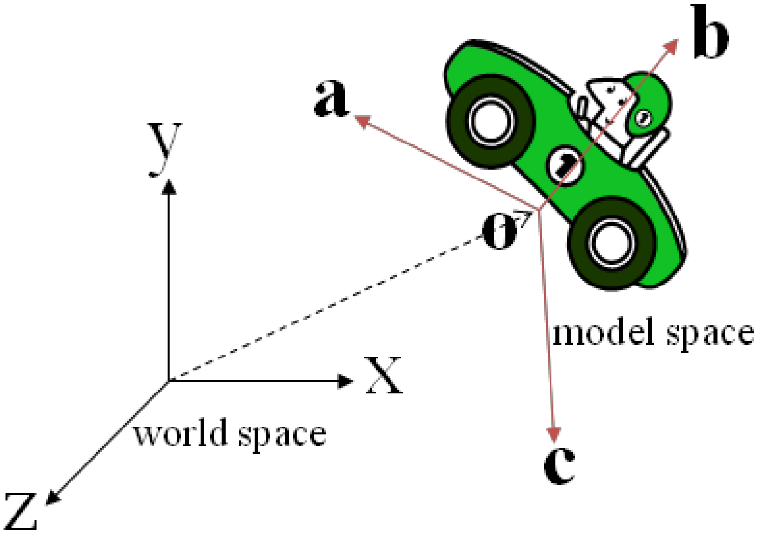
| | |
|---|---|
| Application stage: | swap glVertex to glColor |
| Geometry stage: | remove all light sources |
| Rasterization stage: | Change window size |

---

## Task 2 – Transforms

**a)** **[2p]** Which two classes of transformations are part of Rigid Body Transformations?
**Answer:** translation, rotation

**b)** **[2p]** Give the object's model-to-world matrix.



**Answer:**

$$M_{\text{model-to-world}} = \begin{bmatrix} c_x & b_x & a_x & o_x \\ c_y & b_y & a_y & o_y \\ c_z & b_z & a_z & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Task 3 - Illumination and Visual Appearance

**a)** **[1p]** Which are the 3 components in the real-time illumination model? It is sufficient to just state the names. (**Emission** is often included as the fourth component.)

Answer: ambient, diffuse, specular,

**b)** **[2p**] Compute the reflection ray, **r**, given **n** and **l**, where **n** is the surface normal and **l** is the incoming ray with direction towards the surface.
Answer: $\mathbf{r} = \mathbf{l} - 2*(\hat{\mathbf{n}} \bullet \mathbf{l})\hat{\mathbf{n}}$, (**n** needs to be normalized, **l** does not necessarily)

**c)** **[1p]** Is alpha channel in the color buffer required for correct rendering of transparent objects? Motivate your answer.
**Answer:** No, you state the transparency using the alpha value of the color of the object. The alpha value, *a*, decides the interpolation factor between the source color *c* (the object's color) and the destination color *d* (the color of the pixel in th frame buffer). E.g.: Color = *a*c + (1-*a)d*. The alpha channel in the color buffer does not need to be involved. For correct blending of the transparency, draw the transparent objects in back-to-front order.

**d)** **[1p]** Is the rendering of transparent objects order dependent? Motivate.
Answer: yes, the blending operator is order dependent (unless you have a pure additive or multiplicative blending – but both are used for classic transparency)