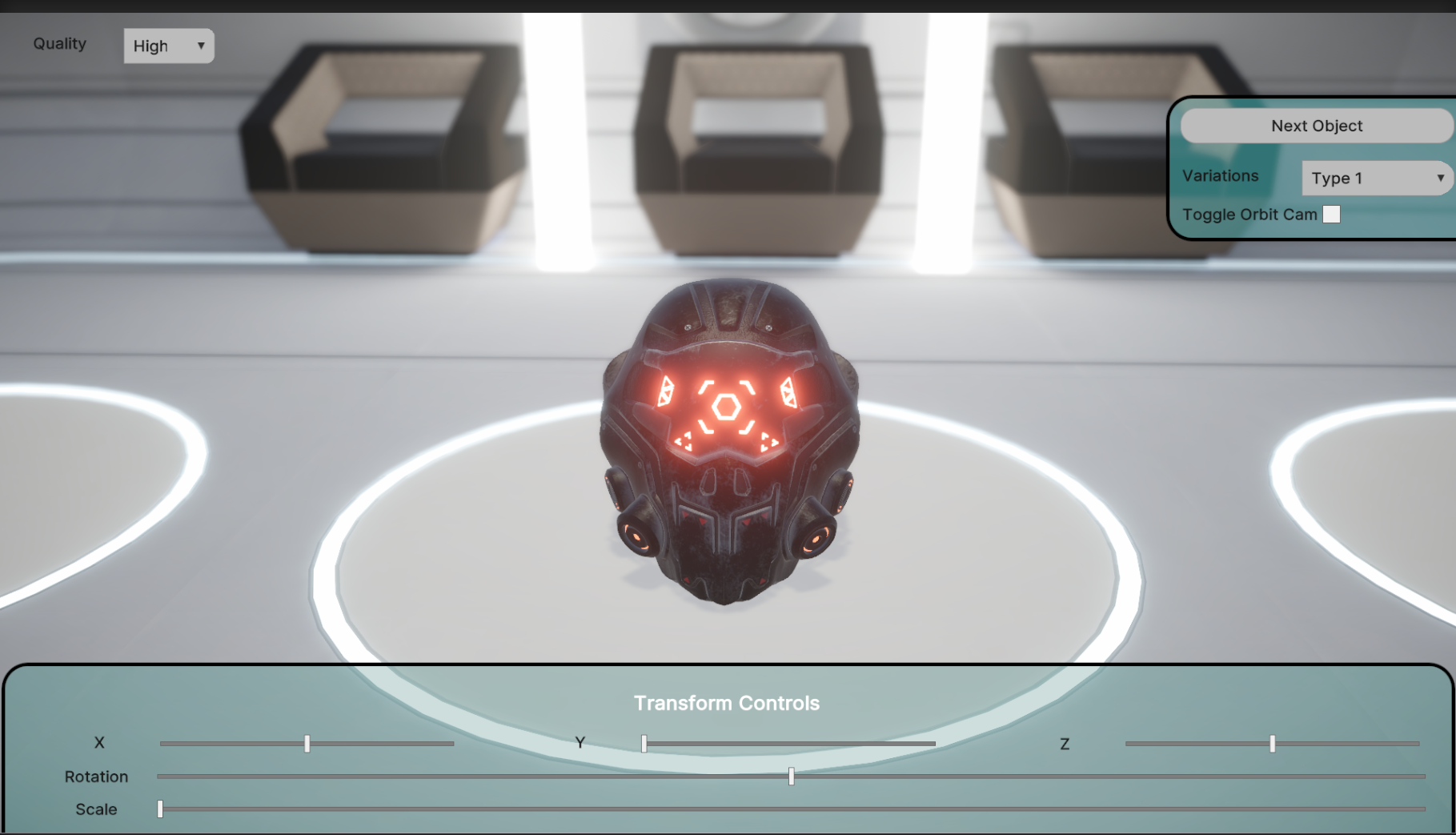
**3D Viewer Documentation**



Contents:

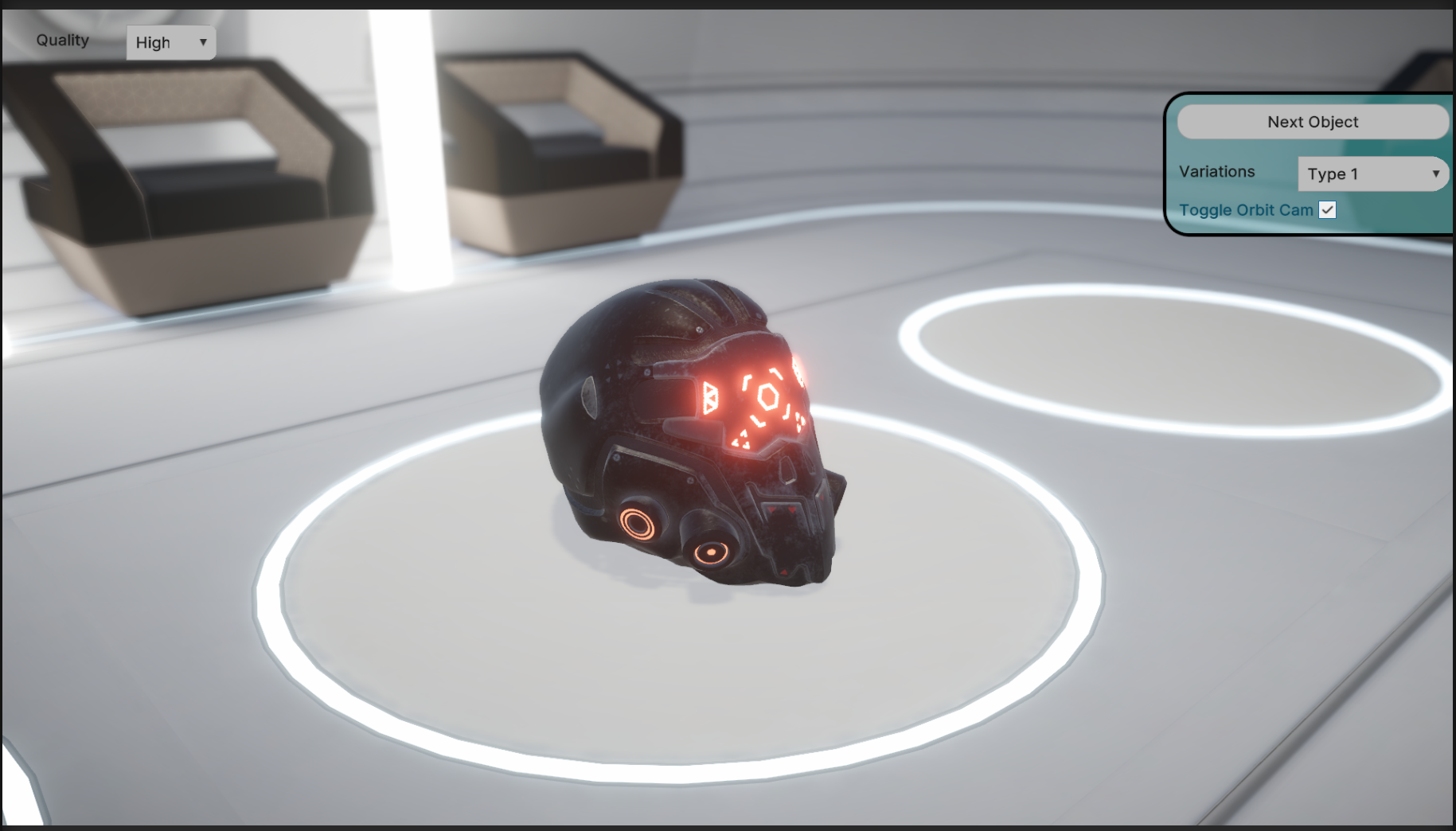
1. Viewer functionality
2. Viewer UI
3. Viewer Quality Levels
4. Code

**Viewer Functionality**

The viewer has a basic functionality of translating around the local axis, rotating around the root axis and scaling around the root axis. The viewer has been preloaded with three 3d models along with 3 different sets of materials for customization of the models. There are 2 modes to using the viewer:

* Orbit Mode
* Fixed Cam Mode

Users can go in an out of orbit mode by using the keyboard shortcut ‘O’ as well along with the in screen button for the same.



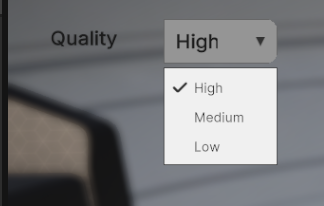
The transform UI controls become visible once the users go into fixed camera mode.

**Viewer UI**

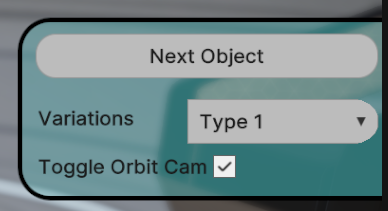
The UI in the viewer is split into 3 parts

1. **Quality Settings Tab**
2. **Model and Camera Controls Tab**
3. **Model Transform Tab**

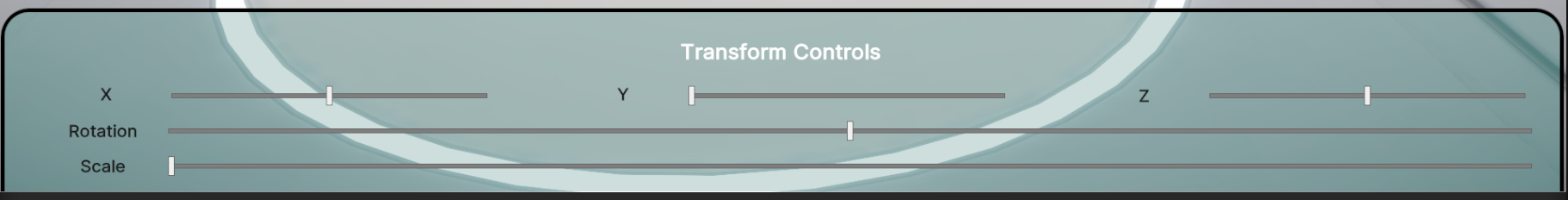
**Quality Settings Tab :** This tab contains a dropdown to toggle between High, Medium and Low quality settings of the viewer. The viewer toggles between different Pipeline Asset Settings and also different post processing volumes.



**Model and Camera Control Tab :** This tab contains a button to toggle between the model assets bundled in the viewer. This tab also contains a drop down to toggle between the variants of the model by changing the materials of the model. Each model has 3 variants to be displayed. The last section of this tab is a toggle between orbit mode and fixed cam mode. The last position of the orbit camera is used as the fixed camera position when it is toggled. You can manually toggle between the 2 by using the keyboard shortcut ‘O’.



**Model Transform Tab :** This tab contains the transform controls for the model. The model can be translated in the local X, Y and Z (clamped) using the respective sliders. The Rotation and Scale can also be changed with their respective sliders.



**Viewer Quality Levels**

There are 3 quality levels to the viewer:

* **High :**

Pipeline Asset

* + AA - 4x
  + HDR enabled

Post Processing

* Bloom
* DoF
* Tonemapping
* Vignette
* **Medium :**

Pipeline Asset

* + AA - 2x
  + HDR enabled

Post Processing

* + Bloom
  + Tonemapping
* **Low :**

Pipeline Asset

* + AA - disabled
  + HDR disabled

Post Processing

* disabled

**Code**

Majority of the functionality has been packed into the UIManager.

public void Initialize(VisualElement root)

This function contains all the initialization of the UI Elements with their respective events.

The UI manager also has functions of ToggleOrbitCam, ResetObject, ResetUI and SwitchOject.

| using UnityEngine; using UnityEngine.UIElements; using UnityEngine.Rendering;   public class UIManager : MonoBehaviour {   //Ui Elements  private Slider X\_Slider, Y\_Slider, Z\_Slider;  private Slider Rot\_Slider;  private Slider Scale\_Slider;   private Button Next\_BTN;  private DropdownField QualityDropdown;  private DropdownField Varients;   private Toggle OrbitCamToggle;   private VisualElement TransformPanel;   public UIDocument UIDocument;   // Scene GameObjects  public GameObject[] SelectedObjectList;  private int selectedOB;  public GameObject RootObject;   private Transform[] InitialObjectTransforms = new Transform[3];  private Transform InitialRootTransform;   public ViewerData Data;   public Cinemachine.CinemachineFreeLook OrbitCam;   public RenderPipelineAsset[] QualityAssets;   public GameObject[] PostVolumes;   private float tempTime;    void Start()  {  //Store initial transforms  for (int i = 0; i < SelectedObjectList.Length; i++)  {  InitialObjectTransforms[i] = SelectedObjectList[i].transform;  }  InitialRootTransform = RootObject.transform;   //Initialize UI  Initialize(UIDocument.rootVisualElement);   //Misc inits  tempTime = Time.time;  QualitySettings.SetQualityLevel(0);  QualitySettings.renderPipeline = QualityAssets[0];  PostVolumes[0].SetActive(true);  PostVolumes[1].SetActive(false);  }   public void Initialize(VisualElement root)  {  //Init & Event handler of translate X   X\_Slider = root.Q<Slider>("XSlider");   X\_Slider.RegisterCallback<ChangeEvent<float>>((evt) =>  {  SelectedObjectList[selectedOB].transform.localPosition = new Vector3(evt.newValue, SelectedObjectList[selectedOB].transform.localPosition.y, SelectedObjectList[selectedOB].transform.localPosition.z);  });   //Init & Event handler of translate Y   Y\_Slider = root.Q<Slider>("YSlider");   Y\_Slider.RegisterCallback<ChangeEvent<float>>((evt) =>  {  SelectedObjectList[selectedOB].transform.localPosition = new Vector3(SelectedObjectList[selectedOB].transform.localPosition.x, evt.newValue, SelectedObjectList[selectedOB].transform.localPosition.z);  });   //Init & Event handler of translate Z   Z\_Slider = root.Q<Slider>("ZSlider");   Z\_Slider.RegisterCallback<ChangeEvent<float>>((evt) =>  {  SelectedObjectList[selectedOB].transform.localPosition = new Vector3(SelectedObjectList[selectedOB].transform.localPosition.x, SelectedObjectList[selectedOB].transform.localPosition.y , evt.newValue);  });   //Init & Event handler of Rotation   Rot\_Slider = root.Q<Slider>("RotSlider");   Rot\_Slider.RegisterCallback<ChangeEvent<float>>((evt) =>  {  RootObject.transform.rotation = Quaternion.Euler(RootObject.transform.rotation.eulerAngles.x, evt.newValue, RootObject.transform.rotation.eulerAngles.z);  });   //Init & Event handler of Scale   Scale\_Slider = root.Q<Slider>("ScaleSlider");   Scale\_Slider.RegisterCallback<ChangeEvent<float>>((evt) =>  {  RootObject.transform.localScale = new Vector3(evt.newValue, evt.newValue, evt.newValue);  });   //Init & Event handler of Material Swap   Varients = root.Q<DropdownField>("DropdownField");   Varients.RegisterCallback<ChangeEvent<string>>((evt) =>  {  switch(evt.newValue)  {  case "Type 1":  if(selectedOB == 0)  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB1\_Set1;  else if (selectedOB == 1)  {  for (int i = 0; i < SelectedObjectList[selectedOB].transform.childCount; i++)  {  SelectedObjectList[selectedOB].transform.GetChild(i).GetComponent<MeshRenderer>().materials = Data.OB2\_Set1;  }  }  else if (selectedOB == 2)  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB3\_Set1;  break;  case "Type 2":  if (selectedOB == 0)  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB1\_Set2;   else if (selectedOB == 1)  {  for (int i = 0; i < SelectedObjectList[selectedOB].transform.childCount; i++)  {  SelectedObjectList[selectedOB].transform.GetChild(i).GetComponent<MeshRenderer>().materials = Data.OB2\_Set2;  }  }  else if (selectedOB == 2)  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB3\_Set2;  break;  case "Type 3":  if (selectedOB == 0)  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB1\_Set3;  else if (selectedOB == 1)  {  for (int i = 0; i < SelectedObjectList[selectedOB].transform.childCount; i++)  {  SelectedObjectList[selectedOB].transform.GetChild(i).GetComponent<MeshRenderer>().materials = Data.OB2\_Set3;  }  }  else if (selectedOB == 2)  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB3\_Set3;  break;  }  });    //Init & Event handler of switching obects   Next\_BTN = root.Q<Button>("Next\_Button");   Next\_BTN.clickable.clicked += () =>  {  ResetObject();  ResetUI();  SwitchObject();  };   TransformPanel = root.Q<VisualElement>("VisualElementTransform");   //Init & Event handler of Orbit Cam Toggle   OrbitCamToggle = root.Q<Toggle>("OrbitCamToggle");   OrbitCamToggle.RegisterCallback<ChangeEvent<bool>>((evt) =>  {  ToggleOrbitCam();  });   //Init & Event handler of Quality Settings   QualityDropdown = root.Q<DropdownField>("Quality");   QualityDropdown.RegisterCallback<ChangeEvent<string>>((evt) =>  {  switch (evt.newValue)  {  case "High":  QualitySettings.SetQualityLevel(0);  QualitySettings.renderPipeline = QualityAssets[0];  PostVolumes[0].SetActive(true);  PostVolumes[1].SetActive(false);  break;  case "Medium":  QualitySettings.SetQualityLevel(1);  QualitySettings.renderPipeline = QualityAssets[1];  PostVolumes[0].SetActive(false);  PostVolumes[1].SetActive(true);   break;  case "Low":  QualitySettings.SetQualityLevel(2);  QualitySettings.renderPipeline = QualityAssets[2];  PostVolumes[0].SetActive(false);  PostVolumes[1].SetActive(false);  break;  }  });  }   //Toggles between Orbit and Fixed Cam  private void ToggleOrbitCam()  {  if (OrbitCam.enabled)  {  OrbitCam.enabled = false;  TransformPanel.visible = true;    }  else  {  OrbitCam.enabled = true;  TransformPanel.visible = false;    }  }   private void ResetObject()  {  //Reset Transforms  RootObject.transform.SetPositionAndRotation(InitialRootTransform.position, InitialRootTransform.rotation);  SelectedObjectList[selectedOB].transform.position = InitialObjectTransforms[selectedOB].position;   //ResetMaterials  switch (selectedOB)  {  case 0:  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB1\_Set1;  break;  case 1:  for (int i = 0; i < SelectedObjectList[selectedOB].transform.childCount; i++)  {  SelectedObjectList[selectedOB].transform.GetChild(i).GetComponent<MeshRenderer>().materials = Data.OB2\_Set1;  }  break;  case 2:  SelectedObjectList[selectedOB].GetComponent<MeshRenderer>().materials = Data.OB3\_Set1;  break;  default:  break;  }  }   //Resets UI Elements  private void ResetUI()  {  X\_Slider.value = 0;  Y\_Slider.value = 0;  Z\_Slider.value = 0;   Rot\_Slider.value = 180;  Scale\_Slider.value = 1;  }   //Switch objects in round robin  private void SwitchObject()  {  SelectedObjectList[selectedOB].gameObject.SetActive(false);  selectedOB++;  if (selectedOB == 3)  selectedOB = 0;  SelectedObjectList[selectedOB].gameObject.SetActive(true);  }    void Update()  {  //Shortkey for toggle between orbit and fixed cam  if (Input.GetKey(KeyCode.O) && Time.time > tempTime)  {  ToggleOrbitCam();  tempTime = Time.time + 1;  }  } } |
| --- |