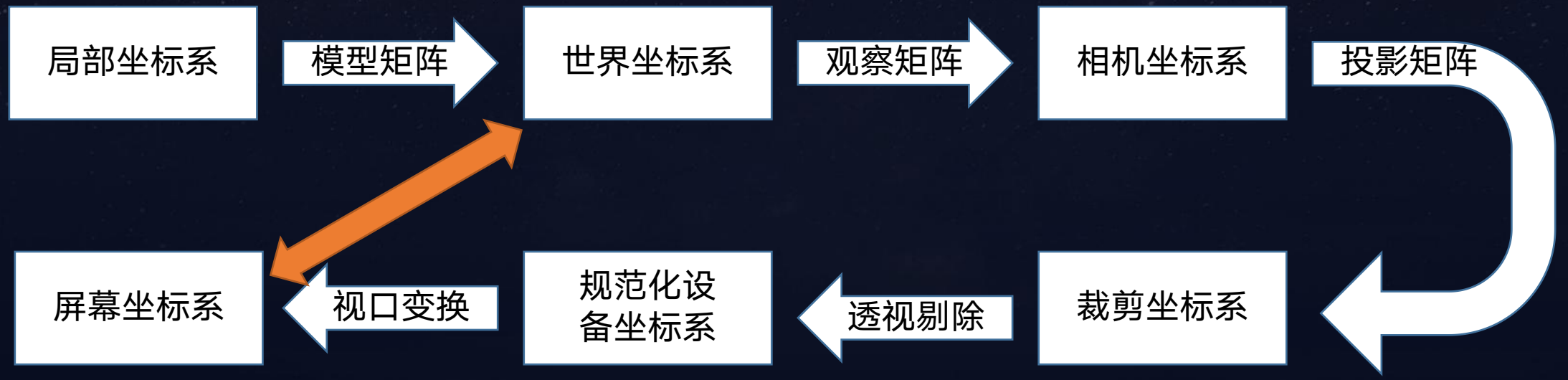
变换流水线


局部坐标系
模型矩阵
世界坐标系
观察矩阵
相机坐标系
投影矩阵
裁剪坐标系
透视剔除
规范化设备坐标系
视口变换
屏幕坐标系

公式推导

世界坐标=>屏幕坐标

局部坐标=>世界坐标=>观察坐标=>裁剪坐标=>标准设备坐标=>屏幕坐标

```
/**
 * 世界坐标系转屏幕坐标系
 * @param {*} worldPosition   世界坐标
 * @param {*} MVPMatrix       MVP变换矩阵
 * @param {*} viewWH          视口宽高
 * @returns
 */
function worldToScreen(worldPosition, MVPMatrix, viewWH) {
  let screenPosition = glMatrix.vec4.create();
  worldPosition = glMatrix.mat4.multiply(screenPosition, MVPMatrix, worldPosition);
  for (let i = 0; i < screenPosition.length ; i++) {
    screenPosition[i] /= screenPosition[screenPosition.length ];
    screenPosition[i] = screenPosition[i] * 0.5 + 0.5;
  }
  screenPosition[0] = screenPosition[0] * viewWH[0];
  screenPosition[1] = viewWH[1] - (screenPosition[1] * viewWH[1]);
  screenPosition = screenPosition.slice(0, -1);
  return screenPosition;
}
```

MVP v = ndc

1、将世界坐标系转换设备坐标系

2、将设备坐标系转换为屏幕坐标系

公式推导

屏幕坐标=>世界坐标

局部坐标<=世界坐标<=观察坐标<=裁剪坐标<=标准设备坐标<=屏幕坐标

```
function screenToWorld(screenPosition, InverseMVPMatrix, viewWH) {
  let worldPosition = glMatrix.vec4.create();
  screenPosition[0] = screenPosition[0] / viewWH[0];
  screenPosition[1] = (viewWH[1] - screenPosition[1]) / viewWH[1];
  screenPosition[2] = screenPosition[2];
  console.log(screenPosition.length);
  for (let i = 0; i < screenPosition.length; i++) {
    screenPosition[i] = screenPosition[i] * 2 - 1;
  }
  worldPosition = glMatrix.mat4.multiply(worldPosition, InverseMVPMatrix, screenPosition);
  worldPosition[0] /= worldPosition[3];
  worldPosition[1] /= worldPosition[3];
  worldPosition[2] /= worldPosition[3];
  worldPosition = worldPosition.slice(0, -1);
  return worldPosition;
}
```

1、将屏幕坐标系转换设备坐标系

2、将设备坐标系转换为世界坐标系

VP v = ndc

v = VP^{-1}ndc