Hidden Surface Removal for GPU: Tutorial

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

About This Document

This document provides an overview of the best approach for exploiting the PlayStation®Vita GPU's ISP (Image Synthesis Processor) unit for reducing fragment processing workload on the shader unit and improving rendering performance. This tutorial presents some common test cases and analyzes each case's performance, highlighting the efficiency that can be obtained from the ISP unit.

This tutorial discusses:

- The main functions of the ISP unit on the front-end of fragment processing
- The performance of the ISP unit for various setups based on the rendering type of primitives (opaque, discard, blend)
- Specific cases
  - Fragment disabling path to the ISP unit (alpha test/discard)
  - Depth replace path to the ISP unit
  - Rendering order
  - Geometry tessellation for better efficiency
- Best Practices for efficient usage of the ISP unit

1 Introduction

The PlayStation®Vita GPU employs Tile Based Deferred Rendering (TBDR). This implies that all the geometry and state data required for rendering a frame is captured and stored per tile after the vertex processing, which facilitates the deferral of fragment processing until after the entire scene has been completely submitted. Because the entire scene is known at the time of rendering, PlayStation®Vita GPU's ISP unit can perfectly apply hidden surface removal to compute only those primitives that are visible in the final scene and perform fragment processing for the corresponding visible fragments.

The fragment processing pipeline for PlayStation®Vita starts with executing the tile command stream from the parameter buffer to send all the geometry binned by tiles to the ISP unit. The unit then performs the hidden surface removal processing, as described in the next section.

Hidden Surface Removal on ISP Unit

The ISP unit is the first stage of the fragment-processing pipeline that executes the tile command stream from the parameter buffer to fetch all the geometry binned by tiles. After a set-up operation, all the triangles in a given tile are processed for hidden surface removal by ISP in two steps. First, depth/stencil tests (always enabled on PlayStation®Vita GPU) are performed on the corresponding fragments by comparing input fragment depth with the value already stored in the depth buffer. The fragments that fail the depth/stencil test are discarded with no further processing. Secondly, hidden surface removal queues the fragments that pass the tests and determines which fragments are visible, successfully ensuring that only they are textured and shaded.

This two step hidden surface removal is a key feature of the PlayStation®Vita GPU architecture. Traditionally, a depth/stencil test is most effective when the primitives are sorted by depth (on CPU) and rendered front-to-back, because this maximizes the fragments that fail the depth/stencil test. However, due to the second step, the sorting of primitives by depth is not necessarily required because fragments that pass the depth/stencil test are later processed and any invisible fragments within the tile are discarded. This ensures that computationally intensive fragment shading operations, such as sampling textures and lighting calculations, are performed only for the valid visible fragments. Occluded pixels are discarded, resulting in high effective fill rates and optimal use of the available shader-unit processing cycles.

Benefits of ISP

The ISP unit has multiple benefits over the traditional forward rendering:

- ISP unit output is independent of the order in which scene geometry were submitted.
- The processing occurs per tile, which allows rendering each tile individually and facilitates high parallelism.
- Applications do not have to resort to external depth sorting a scene, which saves some CPU processing.
- The use of on-chip buffers significantly lowers overall bandwidth requirements when performing depth/stencil tests and visibility testing, even reducing it to zero if the depth/stencil values are no longer needed in future passes or the current scene.

Geometry Categories

Opaque

Opaque objects are those objects that are rendered without any blending/masking or discard/depth replace in the fragment shader. Due to the on-chip hidden surface removal mechanism of

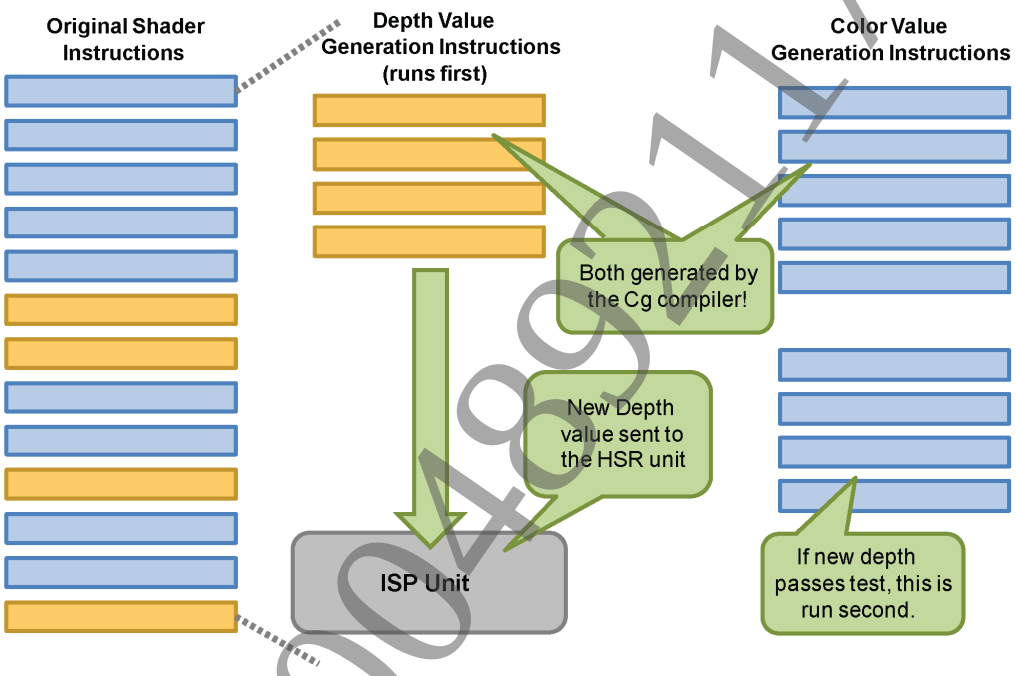
PlayStation®Vita, opaque objects do not require front-to-back sorting (unlike forward rendering). This reduces CPU cost.

Discard / Alpha Test

Geometries that are rendered with fragment shaders using "discard" fall into this category. Normally, for shaders that do not use discard, the depth/stencil test/write is always done before fragment shading even begins. However, when discard is used, the hardware delays the depth write until after the discard operation has passed, to ensure that discarded fragments do not affect the depth buffer. Alpha test on the PlayStation®Vita GPU is only supported through "discard" in fragment shaders.

Because the PlayStation®Vita GPU executes only the instructions that create data for the discard statement that the compiler extracted, it is more efficient than the normal GPU (see Figure 1). However, because the discard statement lowers the efficiency of hidden surface removal, it may negatively affect efficiency.

Figure 1 Inner Instruction Ordering for Shaders Using Discard and Depth Replace



When the PlayStation®Vita GPU renders geometries with "discard", some of the depth buffer optimizations are disabled including limited hidden surface removal support, which affects rendering efficiency.

Depth Replace

Similar to the case where the shaders use discard, when an object is rendered using a fragment shader that outputs the depth values, the efficiency of hidden surface removal goes down; this could negatively affect rendering efficiency. In addition, this operation requires accessing the Z buffer memory, resulting in large bandwidth usage.

Translucent Objects

Translucent objects use fragment shaders that have blending/masking enabled. Note that if you use a color mask other than ARGB, it is treated as a translucent pass type. Pixels are culled when a depth test is failed, even in translucent pass type. In PlayStation®Vita hardware, in addition to programmable blending, the blend/mask operations are exposed by the shader patcher. If the blend info provided when creating the fragment shader is non-NULL, the shader code for implementing the desired blend mode and color mask operation is added.

Additionally, many blend mode cases are render-order dependent and require rendering objects from back to front in order to get correct results. The sorting required for setting the correct render order can be quite CPU intensive, especially when rendering a large number of translucent objects.

Best Practices for Improving HSR Efficiency

The ISP unit can drastically cull occluded fragments, significantly reducing the calculations that the GPU must perform to render the tile. To gain the most benefit from the unit and improve its efficiency, these recommendations should be followed:

You Do Not Need to Sort Objects by Depth

Due to the PlayStation®Vita GPU's ISP unit employing advanced hidden surface removal, it is not necessary for your application to sort opaque objects by depth.

Avoid Depth Test/Discard/Blending

The ISP unit is most efficient when drawing opaque geometries. So you should draw as much of the frame with opaque content as possible and minimize the use of blending, alpha testing, depth replace, and the discard instruction in fragment shaders.

However, if you need to use such geometries, submit these objects separately in the scene after submitting other geometries that do not require it.

Rendering Order

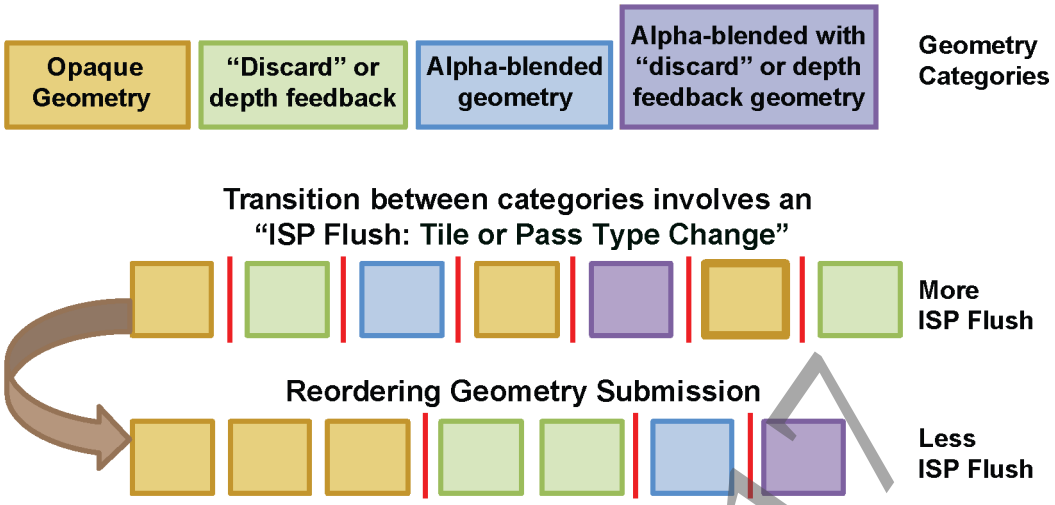
The order in which geometries are submitted for rendering can have a huge impact on the number of state changes required and therefore on performance. Every time geometry is switched from one category to another, the ISP unit may generate a flush. These instances of "ISP Flush: Tile or Pass Type Change" happen independently on each tile, with tiles processed in parallel. Rendering within each tile is done in-order, though globally they can be out-of-order. For opaque geometry, overdraw is minimized within a tile.

If you switch the geometry category on every draw call when rendering a scene, the "ISP Flush: Tile or Pass Type Change" is encountered before each draw call (see Figure 2). This will be quite expensive, reducing the efficiency of the ISP unit and adding to overhead costs. By reordering the submission order of the draw calls to render objects of the same category, you minimize the category changes, resulting in a reduced number of "ISP Flush: Tile or Pass Type Change" and improved performance.

For better performance from the ISP unit, the recommended order to follow when rendering geometries is:

(1) Submit all opaque geometries sorted by render state.
(2) Submit all geometries using discard/depth replace in the shader.
(3) Submit translucent/blended geometries sorted by depth from back-to-front.

Figure 2 Reordering Geometry Submission to Reduce "ISP Flush: Tile or Pass Type Change"



Split Geometry to Separate Alpha Areas

If handled efficiently, alpha test/blending on its own is not overly costly. Because geometries using alpha blending do not benefit from the hidden surface removal feature, it is a performance waste if large sections of the rendered geometry with overlapping polygons actually end up solid on the screen. Thus, to improve performance, only apply blending on the required polygons. It is beneficial to split the geometry so that large opaque sections of the geometry can benefit from hidden surface removal and so that sections using alpha blending can be submitted separately.

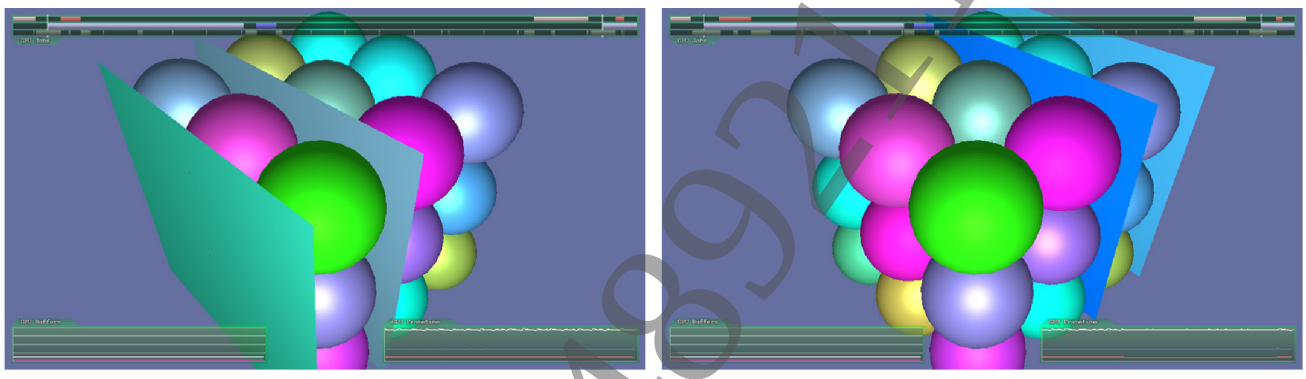
2 Tutorial Walkthrough

This chapter describes a set of five scenarios that have been implemented to test PlayStation®Vita's ISP unit and shows recommended steps that facilitate utilization of the unit's features for optimal efficiency and improved performance. The scenes look at the ISP unit's performance for each rendering mode (opaque, discard, blend) individually, followed by rendering of mixed objects. Each of these scenes is discussed in the following sections.

Opaque Object Scene

This is a simple scene to demonstrate the hidden surface removal feature of the PlayStation®Vita GPU. The scene renders simple opaque overlapping planes and spheres. This scene is rendered either from front-to back (Figure 3a), from back-to-front (Figure 3b), or in random order. However, because the ISP unit culls invisible pixels regardless of the drawing order, there is not much difference in performance.

Figure 3 Opaque Object

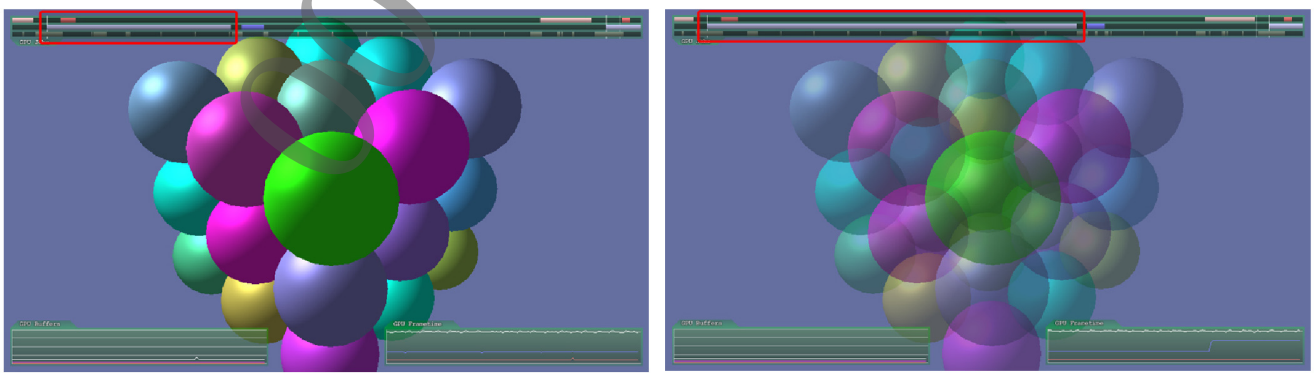


(a) (b)

Blending Scene

This scene renders multiple instances of spheres in a 3D matrix form. The rendering mode for these spheres can be switched from opaque to translucent. As you can see in Figure 4, when blending is enabled, the fragment processing (represented by the blue HUD bars on the top of each figure) is increased.

Figure 4 Demo of Blending's Effect on Performance

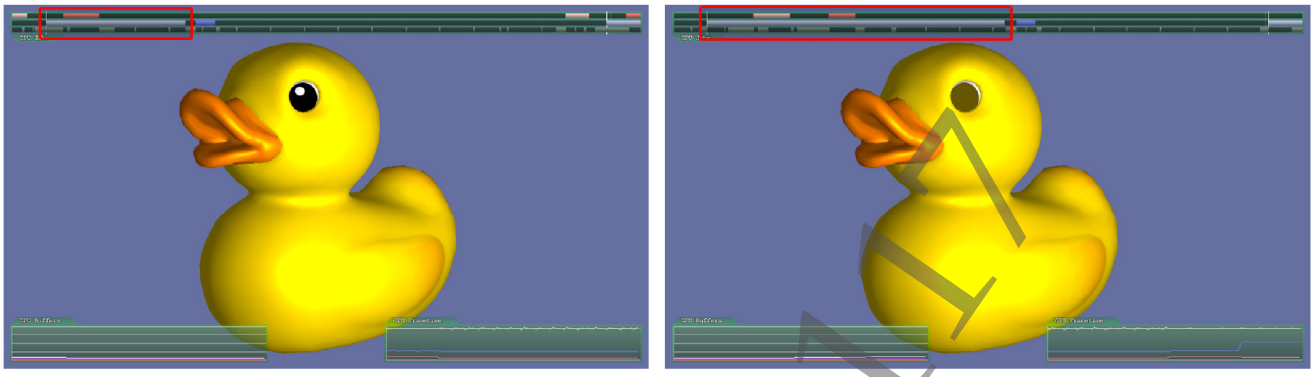


(a) (b)

Discard/Alpha Test Scene

This scene renders the COLLADA™ duck with a texture having low alpha for the eyes. When rendering the duck with a shader using discard for the fragments with low alpha, the eye fragments are not processed. However, using discard increases shader complexity and the scene does not benefit from hidden surface removal, resulting in a higher fragment processing.

Figure 5 Geometries Using Discard Have Limited Hidden Surface Removal, Resulting in Higher Shader-Unit Workload



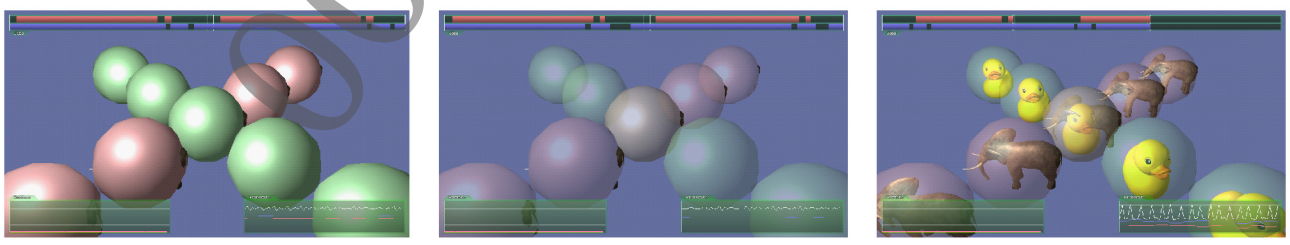
(a) (b)

Rendering Order Scene

This scene renders a few instances of different objects (elephants, ducks, and spheres). These are rendered using different submission orders based on their geometry category. The elephants are always "Opaque", a few instances of COLLADA™ duck are "Opaque", and the others use shader with "Discard". All spheres belong to "Opaque" or, if blending is enabled, the "Blend" category.

The scene includes setting up a number of subcases and testing ISP performance. First, the scene renders the entire scene as opaque (Figure 6a), which implies that the hidden surface removal is applied to primitives corresponding to all geometries. Next the scene is rendered with blend/discard enabled and geometries not sorted (resulting in a high number of "ISP Flush: Tile or Pass Type Change"). This is followed by rendering of geometries sorted by geometry category (reducing the instances of "ISP Flush: Tile or Pass Type Change"), although a random order could produce visually incorrect result – see Figure 6b. After the geometries are sorted by the render category, to achieve better efficiency from the ISP unit the scene is rendered in the order "Opaque", "Discard" and "Blend", with the "Blend" category geometries sorted by depth and rendered back-to-front for correct visual results (Figure 6c).

Figure 6 Different Rendering Order Scenarios



(a) All opaque, rendered randomly

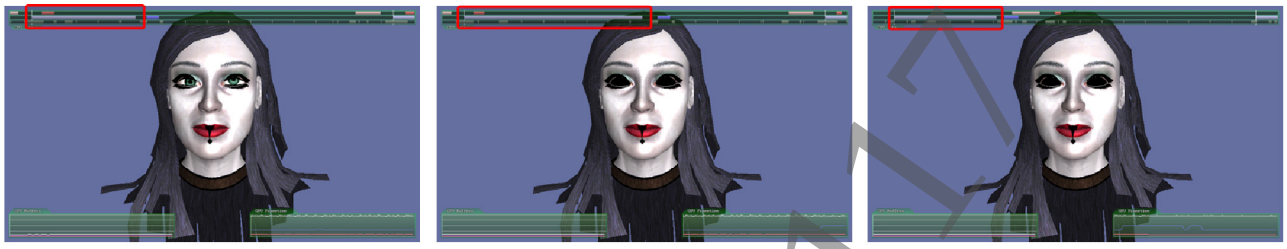
(b) Sorted by geometry category, rendered in random category order

(c) Rendered in order of "opaque", "discard", "blend", sorted by depth

Splitting Geometry Scene

This scene renders a head model. In case (a) in Figure 7, the head and eyes are part of single geometry. The eyes in the head model use alpha blending, which can be enabled from the on-screen menu. In case (b), you can see from the blue HUD bar that the fragment processing increases, resulting in a performance drop when blending is enabled. In case (c), the head and eyes are rendered as two separate geometries: the head is rendered as an opaque mesh, followed by the eyes with translucent texture and blending enabled. The hidden surface removal processing removes the invisible fragments for the separate head, similar to case (a), and improvement in performance is noticed.

Figure 7 Splitting Geometry to Separate Opaque and Alpha Blended Sections, Facilitating Hidden Surface Removal Efficiency and Improving Performance



(a) No Blending

(b) Single geometry with alpha blended eyes

(c) Split geometry with separate alpha blended eyes

This performance variation can be attributed to the following: when the blending mode is set corresponding to the head and eyes as a single geometry, the fragments to be discarded cannot be detected at early stage; thus the ISP unit performs very limited hidden surface removal for the head surface, which increases the processing load on the shader unit because its processing is also performed on fragments corresponding to the occluded geometry. In case (c), when opaque head and translucent eyes are rendered as separate geometries, the head benefits from hidden surface removal, thus reducing the workload on the shader unit.

3 Performance Analysis

This chapter describes some of the measurements made for each of the five scenarios mentioned in the previous chapter. The performance analysis is based on the measurements obtained using the PlayStation®Vita Razor's GPU Performance Analyzer, which supports capture of a number of useful `TRACE` metrics.

The analysis is based on the values for "`TRACE_METRICS = 10`" and "`TRACE_METRICS = 13`" that correspond to the ISP TRACE metrics shown in Table 1. For details on using the metrics, refer to the *Performance Analysis and GPU Debugging* (the user's guide for Razor). The measurements for each of the scenes are shown in Table 2.

Table 1 ISP TRACE Metrics Used

Name	Description
Rasterized Pixels Before Hidden Us	The number of input pixels processed by the ISP over the interval.
Output Pixels	The number of pixels output from the ISP unit and passed for shading and texturing.
Number of ISP Flush: Triangle Count HW Limit Reached	The number of times the ISP is forced to flush because the maximum number of triangles it can handle in a tile has been reached (due to an internal FIFO buffer hardware limit).
Average Tile Time	The average tile processing time – across the entire fragment processing phase – over the selected interval.

Opaque Object Scene

In Table 2, measurements for three subcases are shown for the opaque object scene:

- When the scene is rendered from front-to-back, most of the incoming fragments fail the depth test. The "ISP Flush: Triangle Count HW Limit Reached" value for this case suggests that the ISP unit does not have many polygons in the queue. Approximately 50% of input fragments are occluded and removed.
- In the case of back-to-front rendering, most fragments pass the depth test, resulting in a significant value for "ISP Flush: Triangle Count HW Limit Reached". This implies more processing by hidden surface removal. As in the front-to-back case, a similar number of "input pixels" and "output pixels" are registered, suggesting efficient performance of the hidden surface removal processing by the ISP unit.
- In the final case, even when objects are rendered without sorting, a similar amount of culling is performed. This suggests that CPU sorting is unnecessary for opaque geometries.

Blending Scene

In Table 2, for the opaque subcase we see that all spheres rendered back-to-front benefit from the hidden surface removal feature and that a large percentage of fragments from the tiles processed (~42% of input fragments) are culled. However, when blending is enabled the number of output pixels increases. The small reduction in the output fragments (~7%) can be mainly attributed to the invalid fragments not belonging to the primitive on the output tiles. The increased number of output pixels for the next stage of texturing and shading also implies an increase in average time in processing tiles, as shown in the "Average Tile Time" column.

Discard/Alpha Test Scene

Similar to the alpha blended geometry case, when a geometry is rendered using shader with "discard", the "ISP Flush: Triangle Count HW Limit Reached" is 0 and the percentage of fragments output by the ISP unit is significantly large compared to the opaque case. This again suggests minimal hidden surface removal processing. Additionally, using shader with "discard" also increases the number of shader instructions, as can be seen in the "Shader Instruction Count" column of Table 2.

Table 2 Performance Statistics for ISP TRACE Metrics Under Different Test Scenarios

Test Case	Subcases	Input Pixels Processed by ISP	Output Pixels	ISP Flush: Triangle Count HW Limit Reached	Average Tile Time (ns)	Shader Instruction Count
Hidden Surface Removal (Planes & Spheres)	Rendered front-to-back	1704400	876398	0	58	
	Rendered back-to-front	1719225	880010	10	61	
	Rendered random order	1703508	876500	7	60	
Blend (Sphere Matrix)	With Opaque	1517766	873756	6	55	
	With blending enabled	1406926	1313696	0	121	
Discard (COLLADA™ Duck)	With Opaque	1051111	873756	21	47	34
	With Discard	1344290	1161958	0	75	40
Render Order (Mixed Geometries)	Random Order (Opaque)	1625020	967515	120	67	
	Random Order	1649428	1207679	90	110	
	Sorted by Geometry Category (Random)	1542917	1155416	3	98	
	Sorted by Geometry Category (ODB*)	1643140	1186680	118	117	
	Sorted by Geometry Category (ODB*) and back-to-front for alpha geometries	1633366	1179367	118	114	
Split Geometry (Head Model)	Single Mesh	1393135	887905	31	42	
	Single Mesh with blending	1343485	1162555	3	58	
	Split Geometry with blending	1394466	889889	29	43	

* Opaque/Discard/Blend

Rendering Order Scene

Table 2 shows a number of subcases for this scenario:

- The first subcase is a simple hidden surface removal test where all the objects are rendered as opaque with ~40% fragments being invisible. A very high value for "ISP Flush: Triangle Count HW Limit Reached" here suggests that much hidden surface removal processing is being done.
- The second subcase is the scene rendered with blend/discard enabled and with geometries submitted in random order (no sorting by category). In this case as well, hidden surface removal processing does remove invisible fragments from opaque objects; however, the frequent changes in geometry category results in many instances of "ISP Flush: Tile or Pass Type Change", increasing the processing time for each tile.

- For the third subcase, all the geometries are sorted by geometry category and then all alpha blended spheres are rendered first, followed by discard and then opaque. Sorting geometries by category improves performance. A lower value for "ISP Flush: Triangle Count HW Limit Reached" in this case suggests that most of the invisible pixels for opaque geometries are culled by depth test fail before they ever reach the queueing stage.
- For the fourth subcase, the scene is rendered by geometry category in the following prescribed order: opaque, discard, and blend.
- For the fifth subcase, the scene is rendered as above (fourth subcase) but with blend geometries rendered back-to-front for visually correct results. In terms of performance, in both the fourth and fifth subcases a similar percentage of culled fragments are registered and the processing time is similar.

Note that the processing time per tile in last two subcases is quite high compared with the third subcase. This could be due to the large number of "ISP Flush: Triangle Count HW Limit Reached", indicating that the hidden surface removal performance starts to deteriorate when there is a large number of overlapping primitives per tile.

Splitting Geometry Scene

For this scenario, Table 2 first lists the measurements for the subcase related to rendering the opaque head model. The hidden surface removal processing culls ~36% of the input fragments. Next when blending is enabled the hidden surface removal processing is disabled, with a higher number of fragments passed for shading and texturing. This results in an increase in average tile processing time. However, when the head is split, the opaque section is rendered first via hidden surface removal processing that again culls ~36% of the input fragments, and alpha blending is applied only to the separate eyes. Thus performance in this case does not degrade as a result of using alpha blended geometry.

4 Implementation

This chapter describes the implementation details for various scenes to demonstrate how PlayStation®Vita GPU's ISP unit processing is impacted by the geometry category being rendered.

The sample is located at:

```
SDK/target/samples/sample_code/graphics/tutorial_hidden_surface_removal
```

The sample renders five scenes with each selected from the on-screen menu.

Opaque Object Scene Setup

Select **Test Scene Type** as "HSR" from the on-screen menu.

This scene renders a set of spheres and planes using the simple color shader (`simple_color_f.cg`). All the geometries are rendered with `RenderGeomType` "`OPAQUE`". The scene is first sorted by depth if these geometries need to be rendered front-to-back or back-to-front:

```
// m_pHSRSceneGeoms contains all the geometries for the opaque object scene
// geomCount is the number of geometries for the scene
if((m_hsrTestOrder == FRONT_BACK) || (m_hsrTestOrder == BACK_FRONT))
    quickSortSceneGeom(m_pHSRSceneGeoms, geomCount);
```

Next, depending on the mode for testing, the scene is rendered either front-to-back, back-to-front, or in a non-sorted random order:

```
if((m_hsrTestOrder == FRONT_BACK)) // render front-to-back
{
    for(int k = 0; k < geomCount; k++)
    {
        // render sphere geometries
        if(m_pHSRSceneGeoms[k].geomGroup == RO_SPHERE)
        {
            Sphere *pSphere = (Sphere *)m_pHSRSceneGeoms[k].geomData;
            pSphere->render(m_graphicsData.pContext, viewProjMat,
                            m_eyePosition, m_lightMatrix.getTranslation(),
                            OPAQUE);
        }
    }
}
else // render back-to-front or random
{
    for(int k = geomCount - 1; k >= 0; k--)
    {
       // render sphere geometries as above
    }
}
```

Blending Scene Setup

Select **Test Scene Type** as "Blend" from the on-screen menu. This scene requires setting up blending information and patching it into the fragment shader at initialization.

Blending Info

Different combinations of blending equations and functions can be used depending on the blend modes provided as blending info. The following blending settings are used for the translucent spheres rendered:

```
SceGxmBlendInfo    blendInfo;
blendInfo.colorFunc = SCE_GXM_BLEND_FUNC_ADD;
blendInfo.alphaFunc = SCE_GXM_BLEND_FUNC_ADD;
blendInfo.colorSrc  = SCE_GXM_BLEND_FACTOR_SRC_ALPHA;
blendInfo.colorDst  = SCE_GXM_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA;
blendInfo.alphaSrc  = SCE_GXM_BLEND_FACTOR_ZERO;
blendInfo.alphaDst  = SCE_GXM_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA;
blendInfo.colorMask = SCE_GXM_COLOR_MASK_ALL;
```

Patching Blend Info into Fragment Shader

This blending info is then patched using the shader patcher when the fragment shader is created:

```
returnCode = sceGxmShaderPatcherCreateFragmentProgram
             (  m_pShaderPatcher,
                m_fragmentProgramId,
                fragOutputFormat,
                msaaMode,
                blendInfo,
                sceGxmShaderPatcherGetProgramFromId(m_vertexProgramId),
                &m_pFragmentProgram);
```

For the blended scene, the `RenderGeomType` is set to "`BLEND`" for spheres and rendered using the fragment shader (`basic_color_f.cg`) with patched blending info.

```
m_pSphere->render(m_graphicsData.pContext, viewProjMat, m_eyePosition,
    m_lightMatrix.getTranslation(), BLEND);
```

Discard Scene Setup

Select **Test Scene Type** as "Discard" from the on-screen menu. In this scene, the object needs to be rendered using the fragment shader (`basic_discard_f.cg`) with discard. Alpha test can only be performed through discard, as in the following snippet:

```
half4 main(half4 vTexCoord : TEXCOORD0,
           ...

           uniform sampler2D modelTex : TEXUNIT0): COLOR
{
    half4 textureCol = tex2D(modelTex, vTexCoord.xy);

    if(textureCol.a < 0.5f)
        discard;

    // Per fragment lighting calculations
           ....

    return half4(fragColor, textureCol.a);
}
```

For rendering the COLLADA™ duck with "discard", set `RenderGeomType` to "`DISCARD`".

Rendering Order Scene Setup

Select **Test Scene Type** as "RenderOrder" from the menu. In this scene, initially objects falling into different geometry categories are submitted for rendering in a mixed order. The geometry category is set by setting `RenderGeomType` for the geometry to "`OPAQUE`", "`DISCARD`", or "`BLEND`".

To set the render order of the scene, select the `RenderOrderType` for the scene, which can be done through the on-screen menu by selecting from the "`Scene Render Order`" label. The code for rendering the geometries is identical for all the `RenderOrderType` cases:

```
Matrix4 viewProjMat = m_projectionMatrix * m_viewMatrix;
RenderGeomType geomType = OPAQUE;
const uint32_t geomCount = NUM_RO_GEOMS;

for(int k = 0; k < geomCount; k++)
{
    if(m_enableBlend)  // If enable then the predefined RenderGeomType is set
        geomType =  m_pRenderOrderGeoms[k].geomType;
    if(m_pRenderOrderGeoms[k].geomGroup == RO_SPHERE)
    {
        Sphere *pSphere = (Sphere *)m_pRenderOrderGeoms[k].geomData;
        pSphere->render(m_graphicsData.pContext, viewProjMat, m_eyePosition,
                        m_lightMatrix.getTranslation(), geomType);
    }
    else if(m_pRenderOrderGeoms[k].geomGroup == RO_DUCK  ||
            m_pRenderOrderGeoms[k].geomGroup == RO_ELEPHANT)
    {
        // Render Mesh
        Mesh *pMesh = (Mesh *)m_pRenderOrderGeoms[k].geomData;
        pMesh->render(m_graphicsData.pContext, viewProjMat, m_eyePosition,
                      m_lightMatrix.getTranslation(), geomType);
    }
}
```

If the order is "`TYPE_RANDOM`", different category objects are submitted without sorting the `pRenderOrderGeoms` array, which is out-of-order on initialization. For "`SORTED_RANDOM`" and "`SORTED_ODB`", the scene is sorted by category `RenderGeomType` as:

```
quickSortIterativeSortByRenderType(m_pRenderOrderGeoms, 0, geomCount);
```

and rendered in Blend-to-Opaque and Opaque-to-Blend orders respectively for the two cases. For "`DEPTHSORTED_ODB`" mode, the array sorted by `RenderGeomType` is further sorted by depth for only the `BLEND` category in order to get visually correct results. This is done by calling:

```
quickSortDepthByRenderType(m_pRenderOrderGeoms, BLEND);
```

After the geometries in the array `m_pRenderOrderGeoms` are reorganized as required for the test, the geometries are submitted for rendering.

Splitting Geometry Scene Setup

Select **Test Scene Type** as "Split Mesh" from the menu. To render the single head geometry with translucent eyes, set `RenderGeomType` for the entire geometry to "`BLEND`", which avoids hidden surface removal processing:

```
pSplitGeomMesh[HEAD_EYES]->render(m_graphicsData.pContext, viewProjMatrix,
    m_eyePosition, m_lightPosition, BLEND);
```

When rendering the head and eyes as two separate geometries, the head's draw call is submitted first with its `RenderGeomType` set to "`OPAQUE`" to have it benefit from hidden surface removal processing, and then the draw call for rendering the eyes is submitted with its mode set to "`BLEND`" and with alpha blending enabled:

```
m_pSplitGeomMesh[HEAD]->render(m_graphicsData.pContext, viewProjMatrix,
    m_eyePosition, m_ lightPosition, OPAQUE);
m_pSplitGeomMesh[EYES]->render(m_graphicsData.pContext, viewProjMatrix,
    m_eyePosition, m_ lightPosition, BLEND);
```