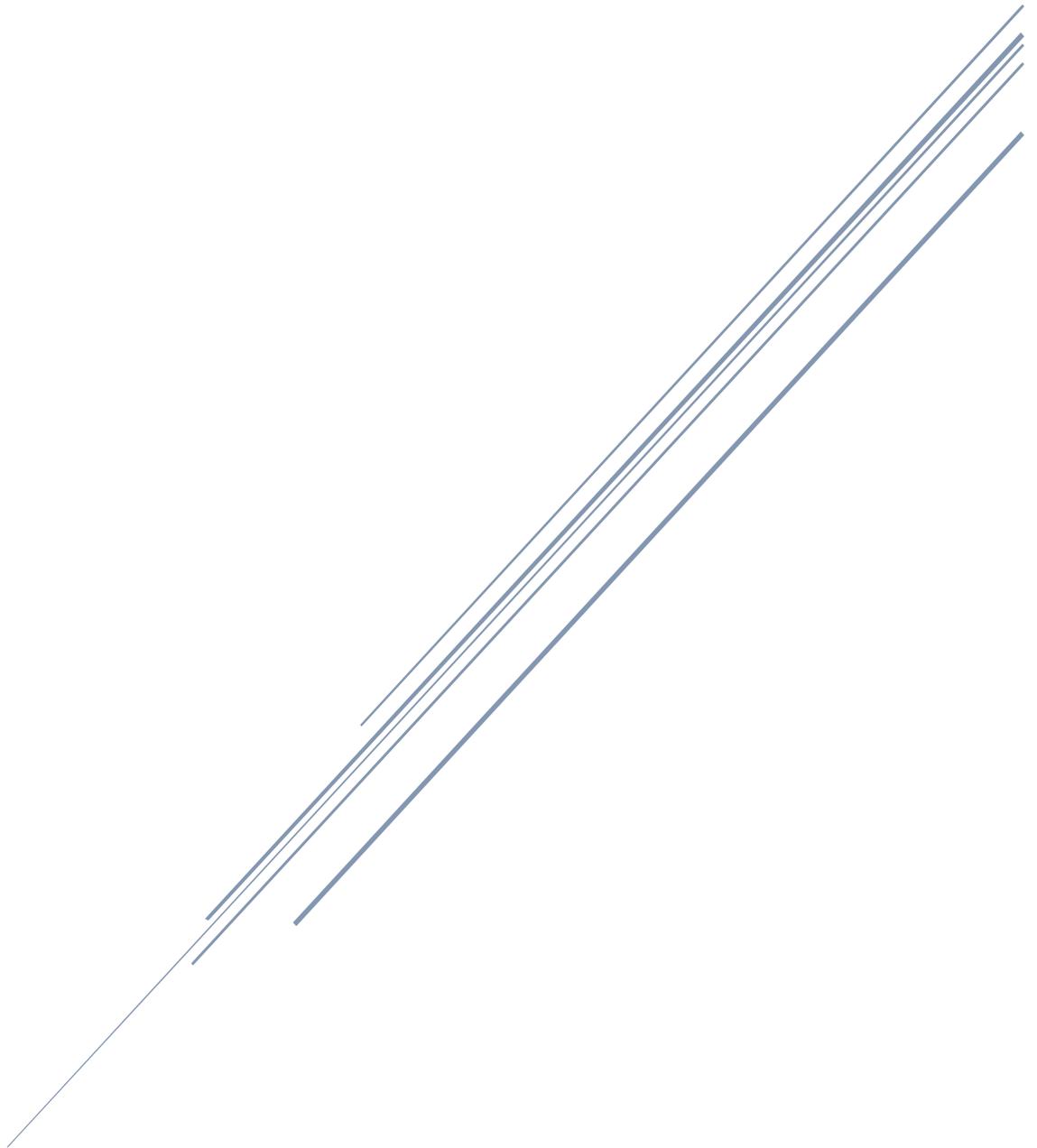


DX9+ GRAPHICS GUIDE

Graphics flow inside D3D9Client



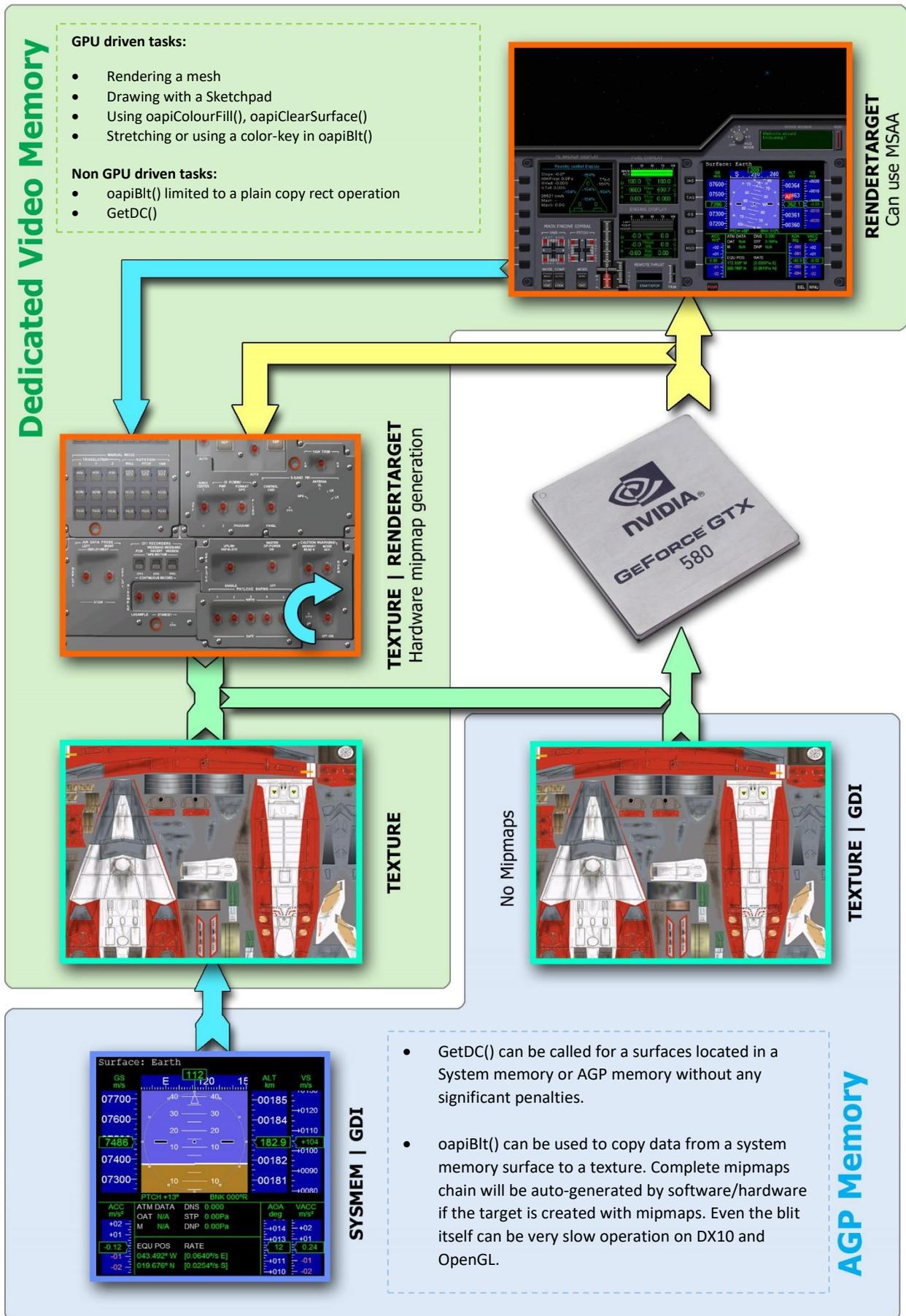


Figure 1- Illustration of surface configuration (Should be DX9, DX10, DX11 and OpenGL compatible)

The GPU

When looking at the graphics flow chart from a previous page you can focus your attention to the GPU. It is the primary execution unit that creates most of the graphics we see in the Orbiter. Any graphics operation that requires processing/writing of a data is executed via programmable GPU. That also includes *oapiBlit*, *oapiClearSurface* and *oapiColourFill* functions. In that case the surface where the graphics operation is targeted for must be a **RENDERTARGET**. The yellow arrow in the flow chart represents that path (*i.e. Output from the GPU*). Also, any surface that is being read by the GPU must be a **TEXTURE**. The green arrow represents this path (*i.e. Input to the GPU*). There exists a “hybrid” surface called **RENDERTARGET | TEXTURE** which can be used by the GPU for both reading and writing. Graphics operations like stretching, blending and color keying are GPU based operations so, the source must be any **TEXTURE** and the target must be any **RENDERTARGET**.

In addition to the GPU there also exists a resource management unit that can transfer data between various graphics resources, this can happen when *oapiBlit* is used to perform a simple/plain copy operation only. Valid source and target surfaces varies based on underlying implementation (e.g. DX11 vs OpenGL). *oapiBlit* between **RENDERTARGET(s)** is well supported in all implementations (*i.e. the blue path*) which also includes a blit from **TEXTURE** to **RENDERTARGET**.

DirectX 11:

CopySubresourceRegion only supports copy; it does not support any stretch, color key, blend, or format conversions.

The “Hybrid” surface and In-surface blitting

The **RENDERTARGET | TEXTURE** “Hybrid” surface can be used for both reading and writing by the GPU. It is a general purpose surface suitable for everything else except for GDI use. It is the most GDI incompatible surface in DX9 and the GDI access is rerouted through a temporary **RENDERTARGET | GDI** surface which also suffers from a poor GDI performance, some lag will be produced as a byproduct. The small curved arrow in a lower-right corner of the “Hybrid” indicates that the surface can be used for in-surface blitting (*i.e. the same surface can be a source and a target at the same time and the rects can overlap*), but that is limited to a simple copy rect operation with *oapiBlit* at a moment. That is not a native DX feature and it is actually done with two copy operations through a temporary surface. The “Hybrid” surface can also have a mipmap chain and, if the surface is modified, the chain is automatically regenerated by hardware whenever the surface is used as a texture in a mesh. Antialiasing is not supported in DX9 while rendering/drawing in the “Hybrid”.

Drawing with GDI

What we have learned so far when working with D3D9Client is that; it’s o.k. to use a GDI to a surfaces that exists in a system memory or AGP memory. Using GDI is fast and doesn’t impact in a render performance. When the **GDI** flag is combined with a **TEXTURE** it becomes a dynamic, which will place the texture in shared AGP memory. It should be noted that a dynamic texture cannot have mipmaps. Underlying client implementation may actually replace a dynamic texture with a dual layer surface having **TEXTURE** main level and **SYSTEM|GDI** sublevel. D3D9Client can provide HDC to any surface but improper use of GDI can cause significant lag and a performance hit.



A rule of thumb is that a surface is either written in by GPU or by GDI but not both. The Sketchpad is using GPU to draw/render graphics. So, use it whenever there is a need to draw something to any **RENDERTARGET**. This also means that the **GDI** flag is potentially incompatible with **RENDERTARGET** and **SKETCHPAD** flags.

Mesh Rendering



Optional Textures Maps supported by the D3D9Client

- **<_norm>** Tangent space normal map. Note: The configuration of this (**_norm**) identifier is ignored if a **_bump** configuration is also found. However, the **_norm** configuration is the preferred one that should be used when you have the choice (see also **_bump**).
- **<_spec>** Specular map controls a specular reflection in per pixel basis. Alpha channel is containing a specular power setting which will be multiplied by 4 inside the client to cover a range 0 to 1024.
- **<_bump>** Bump maps are also supported by the D3D9Client. They are automatically converted into normal maps during loading of bump maps (see note). Valid formats are **<L8>** and **<DXT1>**: Where, **<L8>** is 1-byte per pixel luminance, from **<DXT1>** only green channel is used.
- **<_emis>** Emission map is added to a final color after all other computations has been made.
- **<_refl>** Reflection map in [sRGB] controls material reflectivity in a per pixel basis. Only [RGB] channels from a texture are used. **<_refl>** map is used in combination with **<_rghn>** map.
- **<_rghn>** Monochromatic roughness map specifies material roughness/smoothness. Valid formats are **<L8>** and **<DXT1>** from which only green channel is used. **<_rghn>** is used in combination with **<_refl>** map.
- **<_frsl>** Fresnel map. Which specifies a strength and color of Fresnel reflection. The color is most commonly white but some exceptions may exists. Valid formats are **<L8>** and **<DXT1>**. Note: Fresnel material settings must be defined since the map contains only an intensity.
- **<_transl>** Translucence map controls the material translucency on a per pixel basis. The texture behaves much like a diffuse texture, except that it is illuminated from behind. Illumination is based on the angle of the sun relative to the surface; if the sun is directly behind the surface, the illumination is strong, and if the sun is at a low angle but still behind the surface, the illumination is dim. If the sun begins to illuminate the surface from in front, the translucence effect is no longer visible. Only [RGB] channels from a texture are used.
- **<_transm>** Transmittance map controls the amount of light that passes through a translucent material without being diffused. The effect is very similar to a specular reflection, except that it simulates the bright spot on a semi-transparent material that is directly between the sun and the observer. The [RGBA] channels from a texture are used. The RGB channels provide color and brightness, while the alpha channel changes the size of the bright spot. This is analogous to how the specular map is used.

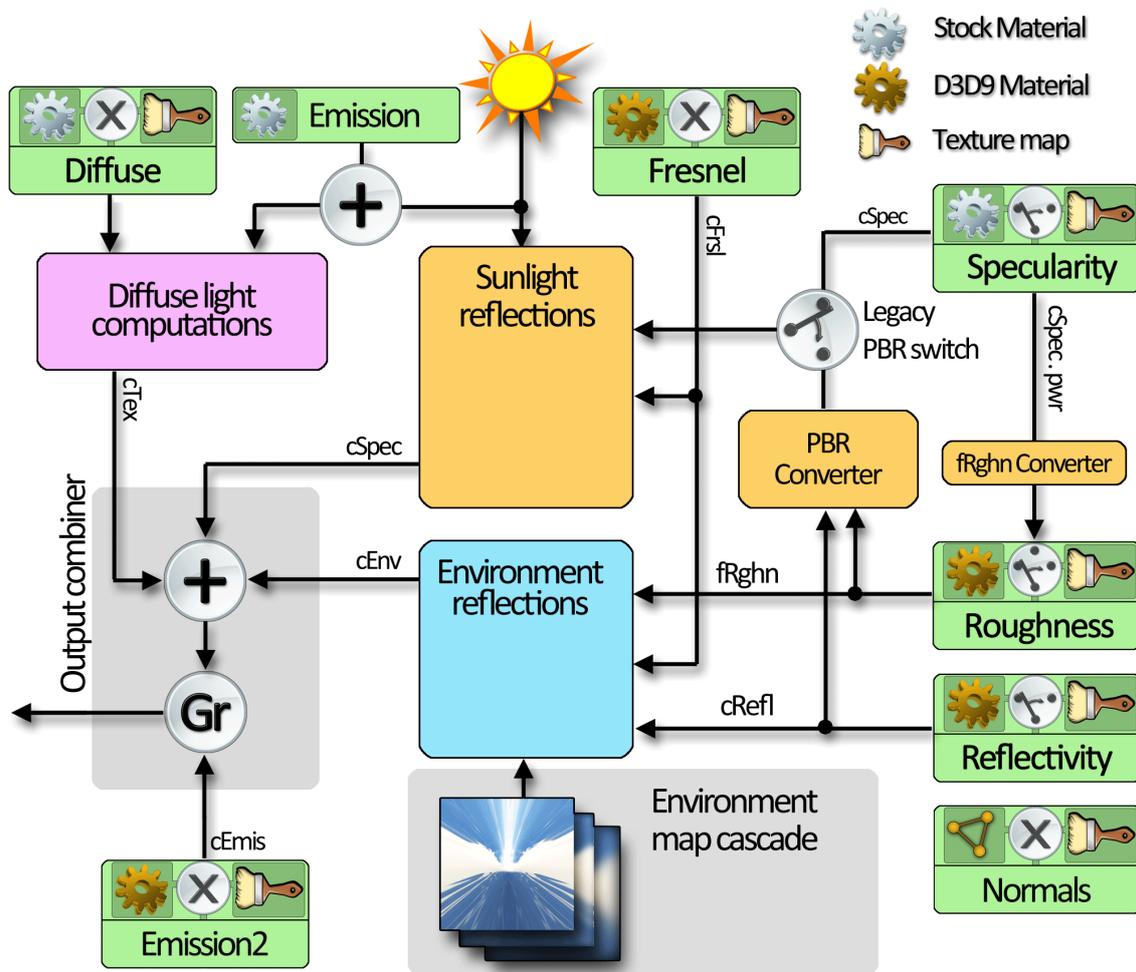


Figure 2- Simplified schematics of the basic graphics pipeline in D3D9Client

Foreword

Orbiter's inline graphics engine doesn't model environment reflections and the specular color only effects in sunlight. Therefore, for the compatibility reasons we cannot really change the behavior of the specular material and how it effects to appearance of a mesh, otherwise we would see unintended and ugly looking reflections here and there. Due to these reasons, the graphics pipeline in D3D9 is split into a two different sections which are sunlight reflections and environment reflections. This configuration also allows us easily to disable environment reflections if needed due to performance issues. The stock (D3D7) and D3D9 specific material properties can be changed/edited with a D3D9Mesh debugger or with [gcMeshMaterial \(\)](#) API function.

Environment reflections

Environment reflections are required to give a materials a proper or better shine. This is optional feature. Meshes and models should be configured so that they render fine with just using the stock materials to maintain a compatibility with DX7. Environment reflections unit is enabled if **all** of the following conditions are satisfied:

- *cRefl* data channel is made valid by applying a reflection texture map or a reflection material property.
- *fRghn* data channel has a valid data from a roughness texture map, a roughness material property or from a specular texture map through *fRghn* converter. That is also the order in which they are used if multiple sources are available at the same time.
- Reflections are enabled from the client configuration dialog.

Legacy / PBR switch

The “legacy PBR switch” is used to select the source of the specular information for the sunlight reflections. The switch is in the “legacy” up position if **any** of the following conditions is true:

- The PBR Converter is unable to operate due to lack of information in *fRghn*, *cRefl* data channels.
- Additional specular texture map is specified.
- The pipeline is specifically told to use the specular material by checking “*Use and Save the property*” from the D3D9Mesh debugger.
- Specular material is set or changed by using *gcMeshMaterial ()* API function

The PBR (Physics based rendering)

To be continued....