EXHIBIT C

U.S. Patent No. 7,633,506

VIZIO / Sigma Designs Products

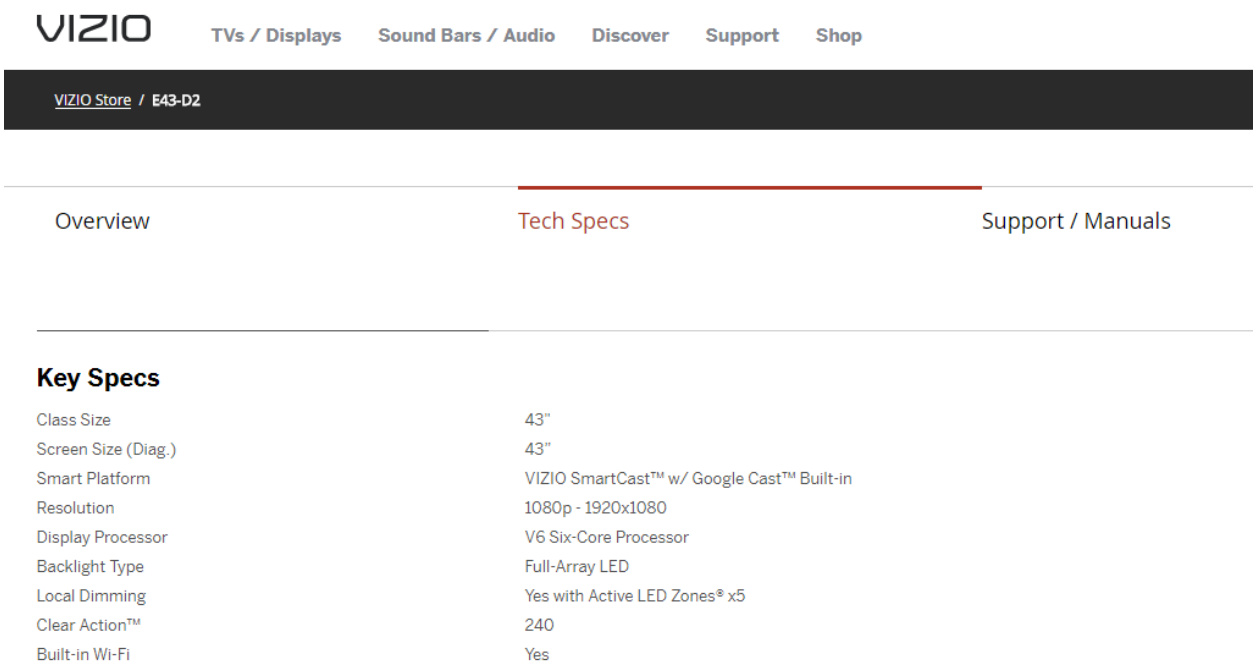
U.S. Patent No. 7,635,506: Claim 1

"1. A graphics chip comprising:"

1. A graphics chip comprising:	The VIZIO television model number E43U-D2 (the “VIZIO Product”) includes a graphics chip.  *See* https://www.vizio.com/e43d2.html. The VIZIO Product includes a VIZIO V6 integrated circuit (the “V6 Integrated Circuit”).

U.S. Patent No. 7,633,506: Claim 1

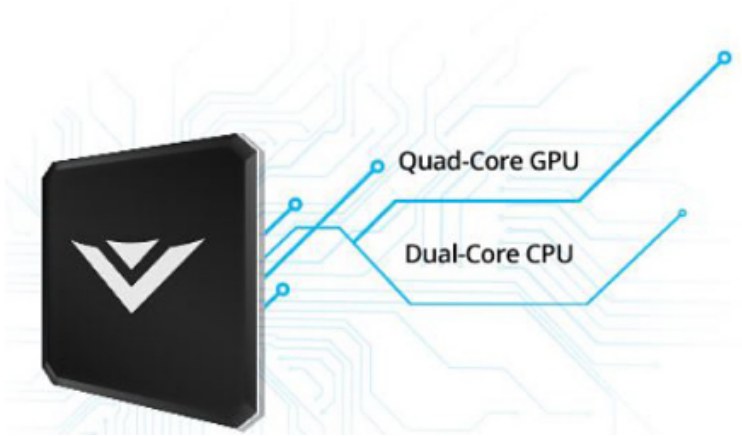
"1. A graphics chip comprising:"



See https://www.vizio.com/e43d2.html.

The V6 Integrated Circuit is powered by a dual core central processing unit and a quad core graphics processing unit.

BE Extra spoke with Carlos Angulo, the senior manager for product marketing at Vizio, about the technology behind these consumer TVs and the impact of 4K/Ultra HD.
BE: Vizio became the first major brand-name to break the $1,000 barrier for a 50-inch 4K TV. Why 4K? Isn't HD good enough? Will 4K be the next 3D, or does 4K have legs?



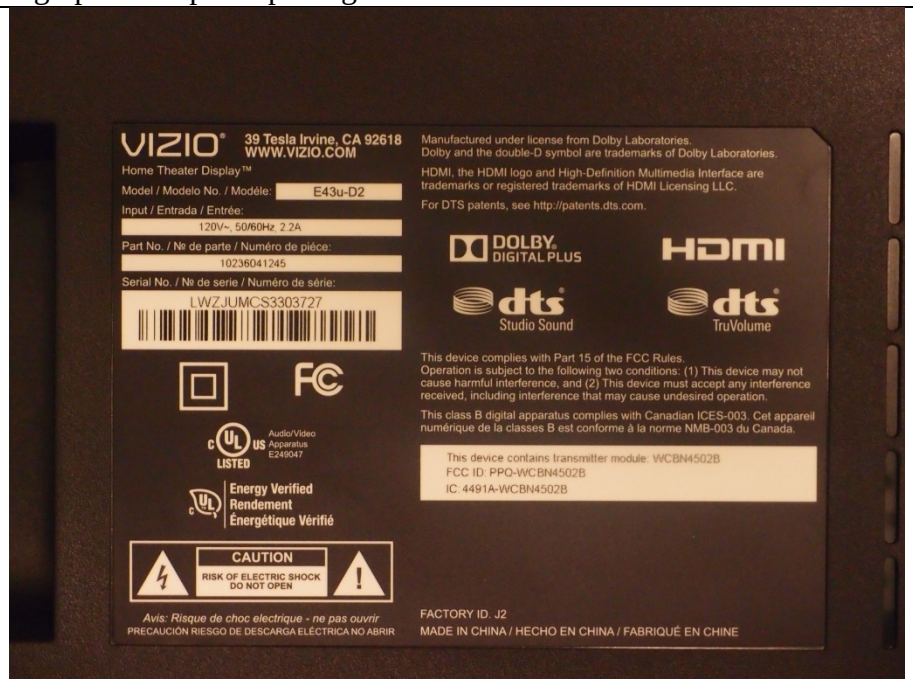
U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"

	See BE Extra, Interview with Carlos Angulo, Senior Manager for Product Marketing at VIZIO, October 9, 2014, http://www.tvtechnology.com/news/0110/ktvs-a-conversation-with-vizios-carlos-angulo/272783. ***BE:*** What are a couple of the technical obstacles to producing a reasonably priced 4K display? **Angulo:** To support the excellent picture quality and Ultra HD experience in our new P-series Ultra HD collection, Vizio engineered a V6 six-core processor. Comprised of a quad-core GPU plus dual-core CPU, the V6 six-core processor delivers maximum speed and advanced graphics processing, enabling an even faster smart TV user experience. To ensure picture quality always remains king, we used the VM50, a dedicated motion and picture-processing engine, to render every image, including Ultra HD content, with incredible detail. P-Series Ultra HD Smart TVs also support the latest standard in streaming thanks to our Internet Apps Plus web interface. *See* BE Extra, Interview with Carlos Angulo, Senior Manager for Product Marketing at VIZIO, October 9, 2014, http://www.tvtechnology.com/news/0110/ktvs-a-conversation-with-vizios-carlos-angulo/272783. The V6 Integrated Circuit is made by Sigma Designs.

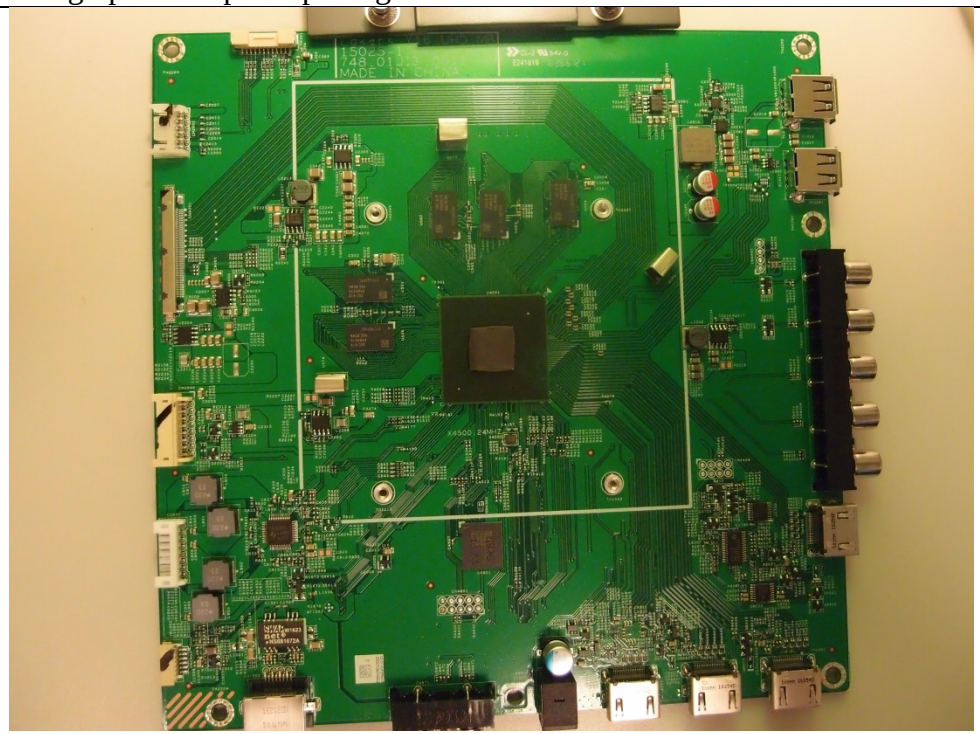
U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"



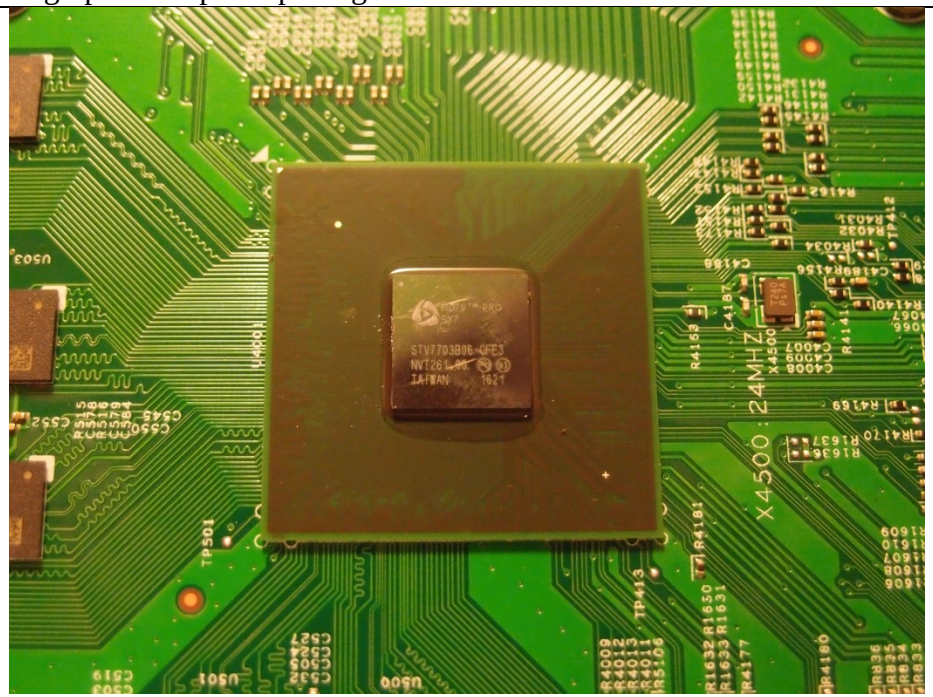
U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"



U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"



See VIZIO E43U-D2 Product Teardown.

Sigma Designs uses the "ARM[] Mali[]-400 MP GPU for their leading range of multimedia SoCs." It is therefore believed that the V6 Integrated Circuit includes an ARM Mali 400 MP4 graphics processing unit (the "Mali GPU").

U.S. Patent No. 7,633,506: Claim 1

"1. A graphics chip comprising:"

Sigma Designs Enhances User Experience in the Digital Home

MILPITAS, CA–(Marketwired – Jan 9, 2014) – Sigma Designs® (NASDAQ: SIGM), a world leader in connected media platforms and the builder of essential consumer semiconductor technologies, recently announced that they have licensed the popular ARM® Mali™-400 MP for their leading range of multimedia SoCs.

The ARM Mali-400 MP is a complete 2D and 3D graphics acceleration platform which provides performance scalable up to 1080p resolution and unrivalled power and bandwidth efficiency. With this license, Sigma Designs will enhance the user experience of the digital home by enabling products with advanced and interactive user interfaces and casual gaming functionality.

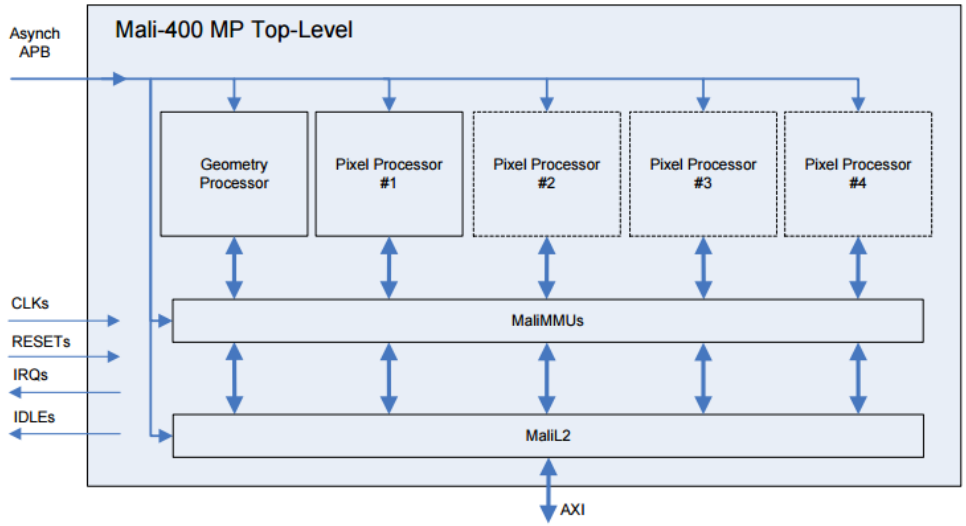
The reach of the digital home continues to grow with a wide portfolio of connected Digital TVs, DMAs and hybrid Set Top Boxes now available. Consumers are now able to have a richer viewing experience with interactive broadcasts, 'over the top' content and rich HTML5 applications. By licensing the ARM Mali-400 MP, Sigma Designs are able to offer the performance level demanded by customers for the advanced GUIs on these devices, as well as the reduced silicon area that is required in cost-sensitive markets. In addition, by pairing the ARM Mali-400 MP with the ARM Cortex®-A9 CPU, Sigma Designs' SoCs will have access to the vibrant Android™ Ecosystem that is optimizing applications for ARM's CPU and GPU technologies.

"The inclusion of ARM technologies in our products will give Sigma Designs an exciting opportunity for enabling new functionality," says Mustafa Ozgen, vice-president and general manager of Sigma Designs' Home Multimedia business unit. "Sigma Designs' SoCs will offer superior computing and graphics performance in addition to their already acclaimed superior multimedia performance and the ability to scale from entry-level set-top boxes to high-end digital TVs."

"By licensing the ARM Mali-400 MP, Sigma Designs can help consumers and their families discover stunning visual displays and fresh digital experiences at home, from highly-accessible gaming to easy-to-use yet advanced and intuitive new interfaces," said Trina Watt, Vice President, Solutions Marketing, ARM. "ARM is delighted to help Sigma Designs bring the magic of ARM Mali graphics and ARM Cortex processor technologies to many more of the world's households."

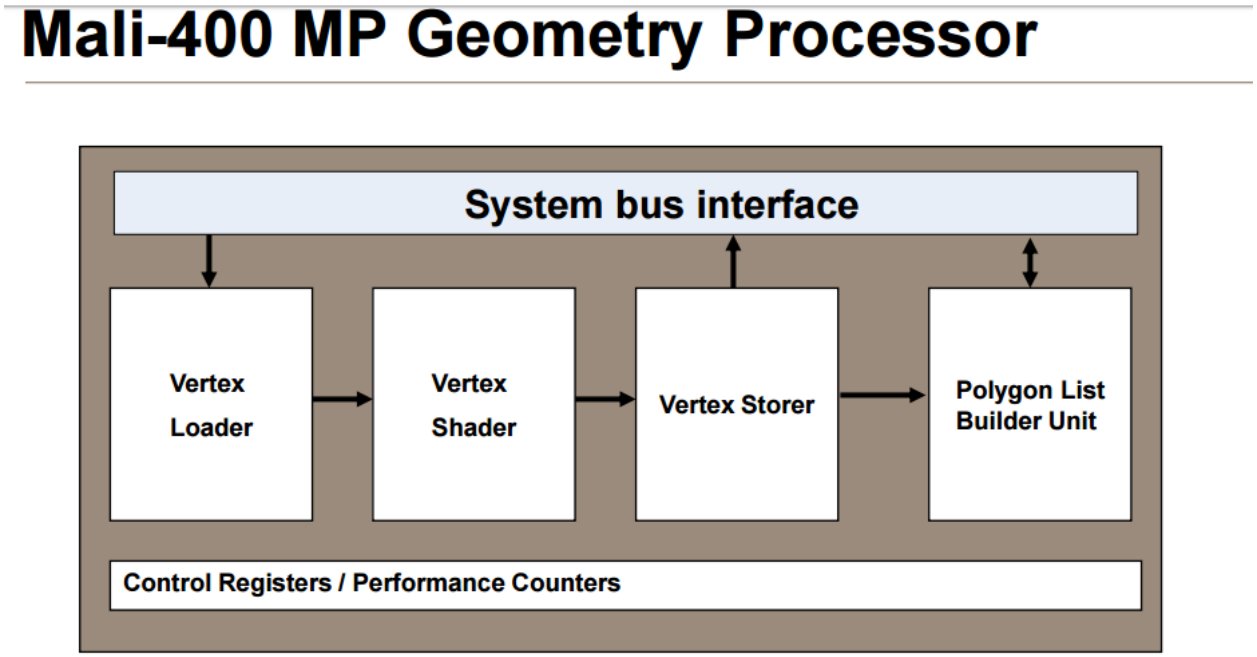
See http://malideveloper.arm.com/news/sigma-designs-enhances-user-experience-in-the-digital-home/.

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;	The VIZIO Product includes a front-end in the graphics chip configured to receive one or more graphics instructions and to output geometry. For example, the Mali GPU includes a Geometry Processor (the "Geometry Processor"). **Mali 400-MP Top Level Architecture**  ■ Scalable pixel performance ■ 1-4 rasterizer cores ■ 32K-128K L2 cache mali Bringing Visual Entertainment to Life 8 ARM *See* Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.8. The Geometry Processor includes a Vertex Loader, Vertex Shader, Vertex Storer, and Polygon List Builder Unit.

U.S. Patent No. 7,633,506: Claim 1

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"



See Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.10.

The Mali GPU includes a front-end configured to receive one or more graphics instructions. For example, "Mali GPUs use data structures and hardware functional blocks[.]" Moreover, "shaders specify the vertex and fragment processing operations."

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

1.6.1 About the OpenGL ES 3.x pipeline

Mali GPUs use data structures and hardware functional blocks to implement the OpenGL ES graphics pipeline.

In the OpenGL ES 3.x pipeline, the shaders specify the vertex and fragment processing operations. The application must provide a pair of shaders for each draw call. A vertex shader defines the vertex processing operations and a fragment shader defines the fragment processing operations. The vertex shader is executed once per vertex, and the fragment shader is executed once per fragment.

Most of the semantics that are associated with data flowing through the pipeline are abstracted into the following special variables that are declared in the shaders:

- Generic vertex attributes. Generic vertex attributes replace all vertex data, such as position, normal vector, texture coordinates, and colors.
- Varying variables. All outputs from the vertex shader, except for position and point size, are abstracted into varying variables. These variables are interpolated across the primitive and are available to the fragment shader.
- Uniform variables. All global states that are required by vertex and fragment processing, such as transformation matrices, light positions, material properties, texture stage constants, and texture bindings, are abstracted into uniform variables. The application sets the values of these variables.

The following figure shows a simplified OpenGL ES graphics pipeline:

See http://malideveloper.arm.com/downloads/OpenGLES3.x/arm_mali_gpu_opengl_es_3-x_developer_guide_en.pdf.

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

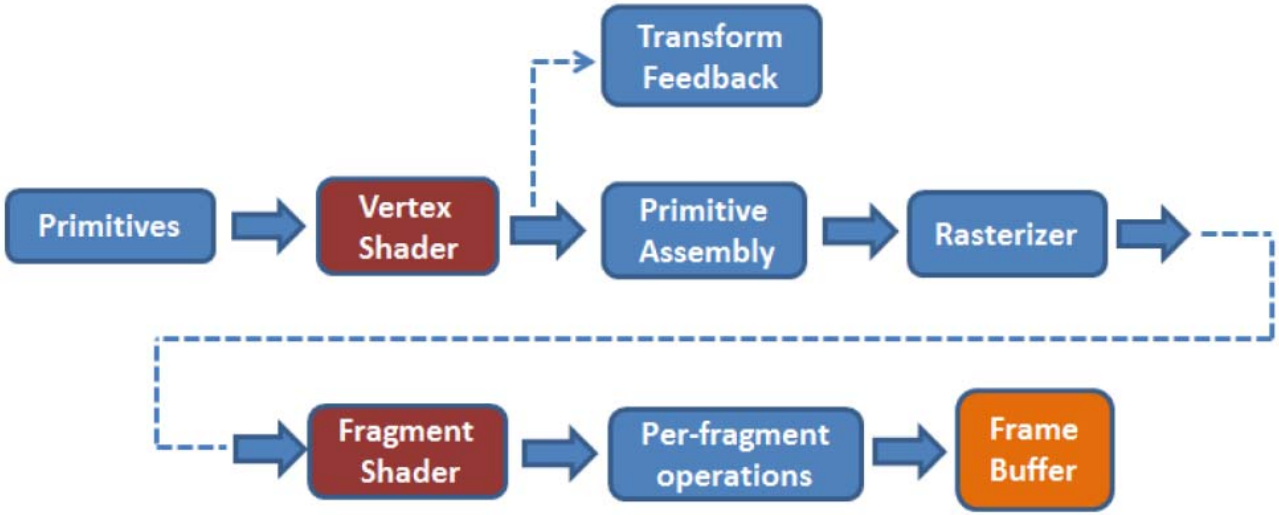


Figure 6-2 OpenGL ES 3.0 Programmable Pipeline

Primitives

In the primitives stage the pipeline operates on the geometric primitives described by vertices, points, lines and polygons.

Vertex Shader

The vertex shader implements a general-purpose programmable method for operating on vertices. The vertex shader transforms and lights vertices.

U.S. Patent No. 7,633,506: Claim 1

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

Primitive assembly

In primitive assembly the vertices are assembled into geometric primitives. The resulting primitives are clipped to a clipping volume and sent to the rasterizer.

Rasterization

Output values from the vertex shader are calculated for every generated fragment. This process is known as interpolation. During rasterization, the primitives are converted into a set of two-dimensional fragments that are then sent to the fragment shader.

Transform feedback

Transform feedback, enables writing selective writing to an output buffer that the vertex shader outputs and is later sent back to the vertex shader. This feature is not exposed by Unity but it is used internally, for example, to optimize the skinning of characters.

Fragment shader

The fragment shader implements a general-purpose programmable method for operating on fragments before they are sent to the next stage.

Per-fragment operations

In Per-fragment operations several functions and tests are applied on each fragment: pixel ownership test, scissor test, stencil and depth tests, blending and dithering. As a result of this per-fragment stage either the fragment is discarded or the fragment color, depth or stencil value is written to the frame buffer in screen coordinates.

See "ARM Guide to Unity" Version 2.1, available at http://infocenter.arm.com/help/topic/com.arm.doc.100140_0201_00_en/arm_guide_to_unity_enhancing_your_mobile_games_100140_0201_00_en.pdf, at pps. 6-75, 76.

The front-end outputs a geometry. For example, "[t]he Mali GPU generates primitives starting from the vertices." Moreover, the "vertex processor…[a]ssembles vertices of graphics primitives" and "[b]uilds polygon lists." Furthermore, "[t]he output of vertex processing includes…[t]he position of the vertex in the output frambuffer" and "[a]dditional data, such as the color of the vertex after lighting calculations."

"a front-end in the graphics chip configured to receive one or more graphics instructions and to output a geometry;"

1.6.2 Primitive assembly

The Mali GPU generates primitives starting from the vertices.

A point contains one vertex, a line contains two vertices, a triangle contains three vertices. Vertices can be shared between multiple primitives, depending on the draw mode. If geometry or tessellation shaders are present, then vertices can generate a variable number of primitives.

1.6.3 Vertex processing

The vertex data provided by the application is read one vertex at a time, and the shader core runs a vertex shader program for each vertex.

This shader program performs:

- Lighting.
- Transforms.
- Viewport transformation.
- Perspective transformation.

The shader core or vertex processor also perform the following processing:

- Assembles vertices of graphics primitives.
- Builds polygon lists.

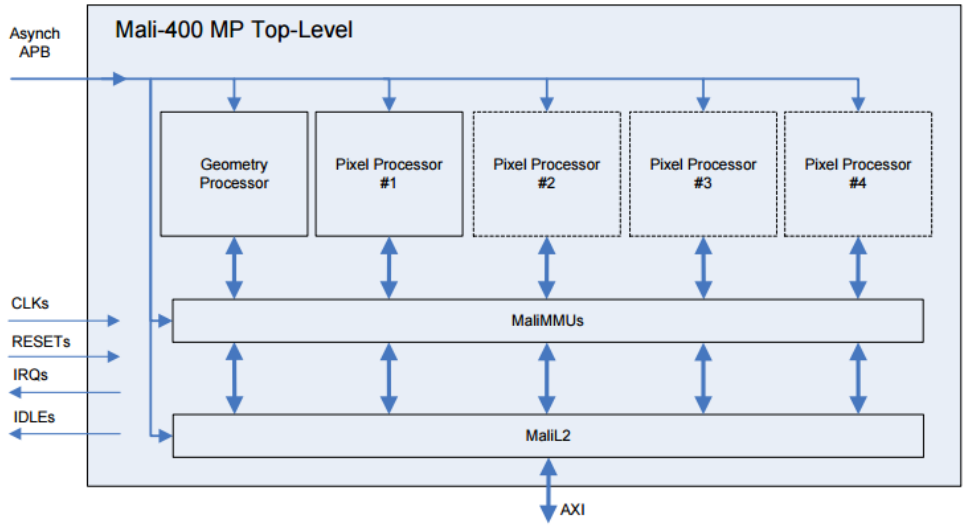
The output from vertex processing includes:

- The position of the vertex in the output framebuffer.
- Additional data, such as the color of the vertex after lighting calculations.

See http://malideveloper.arm.com/downloads/OpenGLES3.x/arm_mali_gpu_opengl_es_3-x_developer_guide_en.pdf.

U.S. Patent No. 7,633,506: Claim 1

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;	The VIZIO Product includes a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer. For example, the Mali GPU includes a back-end in the graphics chip configured to receive said geometry and to process said geometry. For example, the Mali GPU includes four fragment processors (the "Fragment Processors"), which receive processed primitives from the Geometry Processor. **Mali 400-MP Top Level Architecture**  ■ Scalable pixel performance ■ 1-4 rasterizer cores ■ 32K-128K L2 cache mali Bringing Visual Entertainment to Life 8 ARM *See* Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.8.

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

	6.1.9 Fragment Shaders The fragment shader is the graphics pipeline stage after primitive rasterization. For each sample of the pixels covered by a primitive, a fragment is generated. The fragment shader code is executed for each generated fragment. There are many more fragments than vertices so you must take care about the number of operations performed in the fragment shader. In the fragment shader you can access the fragment coordinates in the windows space among other values that contains all interpolated per-vertex output values from the vertex shader. In the shader example in *6.1.2 Shader structure* on page 6-72, the fragment shader receives the interpolated texture coordinates from the vertex shader and performs a texture lookup to obtain the color at these coordinates. It combines this color with the ambient color to produce the final output color. From the declaration of the fragment shader `float4 frag(vertexOutput input) : COLOR` it is clear that it is expected to produce the fragment color. The fragment shader is where you do the operations to achieve the required effect. This ultimately consists of assigning the correct color to a fragment. *See* "ARM Guide to Unity" Version 2.1, available at http://infocenter.arm.com/help/topic/com.arm.doc.100140_0201_00_en/arm_guide_to_unity_enhancing_your_mobile_games_100140_0201_00_en.pdf, at p. 6-79.

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

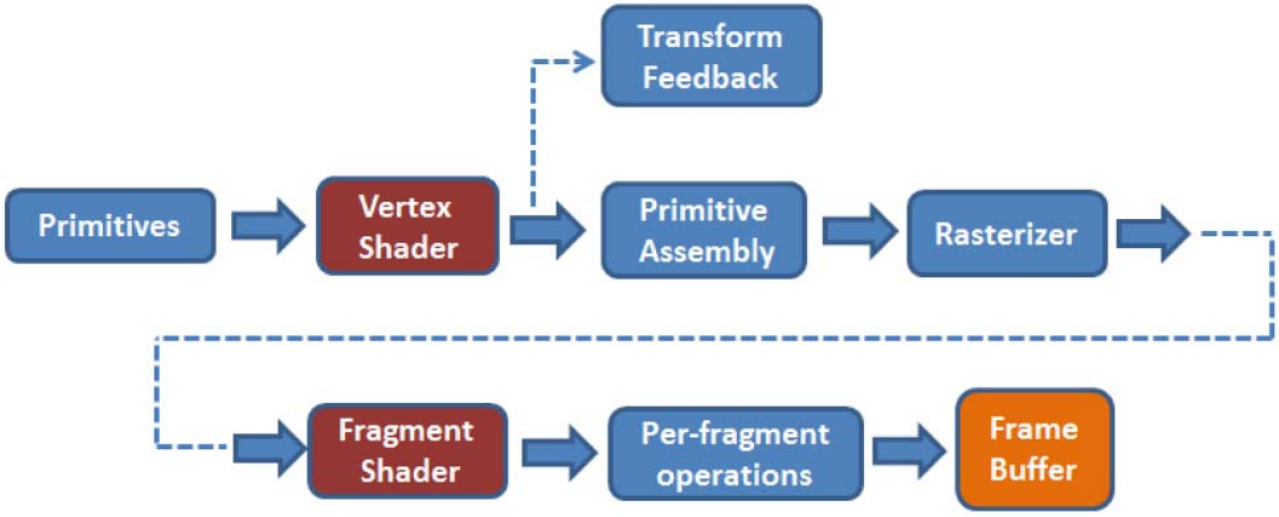


Figure 6-2 OpenGL ES 3.0 Programmable Pipeline

Primitives

In the primitives stage the pipeline operates on the geometric primitives described by vertices, points, lines and polygons.

Vertex Shader

The vertex shader implements a general-purpose programmable method for operating on vertices. The vertex shader transforms and lights vertices.

U.S. Patent No. 7,633,506: Claim 1

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

Primitive assembly

In primitive assembly the vertices are assembled into geometric primitives. The resulting primitives are clipped to a clipping volume and sent to the rasterizer.

Rasterization

Output values from the vertex shader are calculated for every generated fragment. This process is known as interpolation. During rasterization, the primitives are converted into a set of two-dimensional fragments that are then sent to the fragment shader.

Transform feedback

Transform feedback, enables writing selective writing to an output buffer that the vertex shader outputs and is later sent back to the vertex shader. This feature is not exposed by Unity but it is used internally, for example, to optimize the skinning of characters.

Fragment shader

The fragment shader implements a general-purpose programmable method for operating on fragments before they are sent to the next stage.

Per-fragment operations

In Per-fragment operations several functions and tests are applied on each fragment: pixel ownership test, scissor test, stencil and depth tests, blending and dithering. As a result of this per-fragment stage either the fragment is discarded or the fragment color, depth or stencil value is written to the frame buffer in screen coordinates.

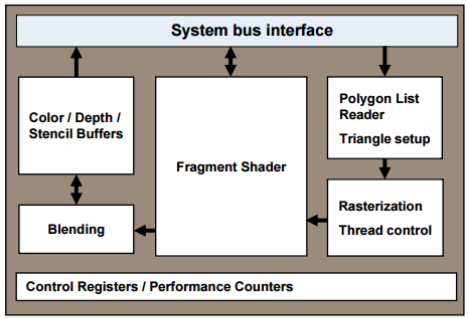
See "ARM Guide to Unity" Version 2.1, available at http://infocenter.arm.com/help/topic/com.arm.doc.100140_0201_00_en/arm_guide_to_unity_enhancing_your_mobile_games_100140_0201_00_en.pdf, at pps. 6-75, 76.

The geometry is processed into one or more final pixels to be placed in a frame buffer. For example, the Fragment Processor includes Color / Depth / Stencil Buffers, a Blending, Fragment Shader, Polygon List Reader, Triangle Setup, Rasterization, and Tile Buffer circuitry. Moreover, the Mali GPU "[r]asterizes [polygons] into an on chip 16X16 tile buffer" before writing the final pixels to an off-chip frame buffer.

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

Mali-400 MP Pixel Processor

- Fragment Shader
  - 128-thread barrel processor
  - Fully general control flow
  - VLIW ISA, tuned for graphics
  - One texture sample per clock



- Key Features
  - 16x16 on-chip tile buffer
  - Renders one pixel per clock: 275M pix/sec @ 275 MHz
  - No penalty for 4x MSAA
  - No penalty for blending
  - 4x or 16x MSAA resolve on output
  - OpenGL ES 2.0 states handled without state dependent shaders

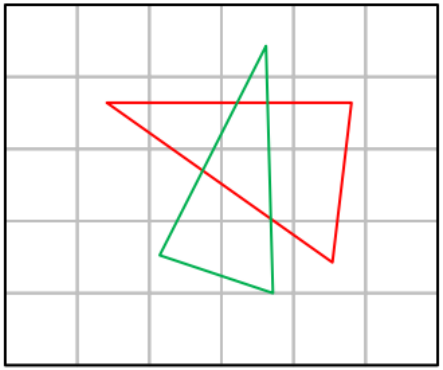
mali Bringing Visual Entertainment to Life 11 ARM

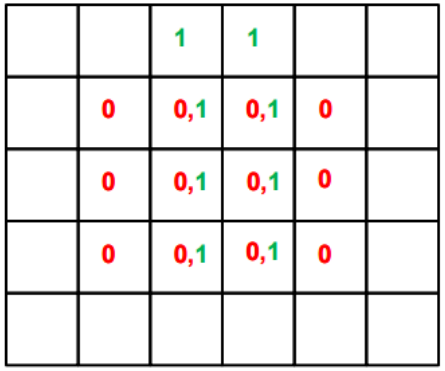
See Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.11.

U.S. Patent No. 7,633,506: Claim 1

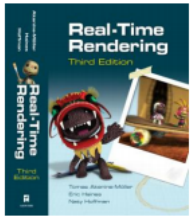
"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

Mali Tile-Based Rendering





- Reduces off-chip framebuffer bandwidth
  - Rasterize into on-chip 16x16 tile buffer
  - Z, stencil, MSAA samples never go off-chip
  - Tradeoff against increased geometry bandwidth
- Details: see *Real-Time Rendering, 3rd ed.*



mali Bringing Visual Entertainment to Life 9 ARM

See Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.9.

U.S. Patent No. 7,633,506: Claim 1

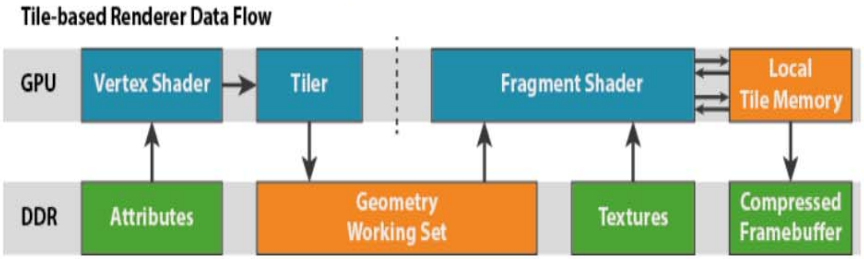
"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

The Mali Approach

The Mali GPU family takes a very different approach, commonly called tile-based rendering, designed to minimize the amount of power hungry external memory accesses which are needed during rendering. As described in the first blog in this series, Mali uses a distinct two-pass rendering algorithm for each render target. It first executes all of the geometry processing, and then executes all of the fragment processing. During the geometry processing stage, Mali GPUs break up the screen into small 16x16 pixel tiles and construct a list of which rendering primitives are present in each tile. When the GPU fragment shading step runs, each shader core processes one 16x16 pixel tile at a time, rendering it to completion before starting the next one. For tile-based architectures the algorithm equates to:

```
01.    foreach( tile )
02.         foreach( primitive in tile )
03.              foreach( fragment in primitive in tile )
04.                   render fragment
05.
```

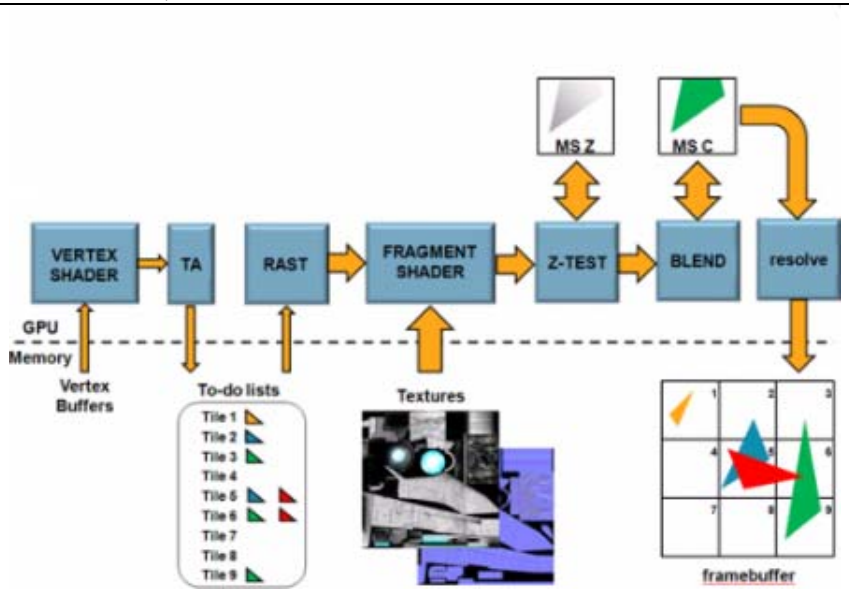
As a 16x16 tile is only a small fraction of the total screen area it is possible to keep the entire working set (color, depth, and stencil) for a whole tile in a fast RAM which is tightly coupled with the GPU shader core.



See https://community.arm.com/groups/arm-mali-graphics/blog/2014/02/20/the-mali-gpu-an-abstract-machine-part-2.

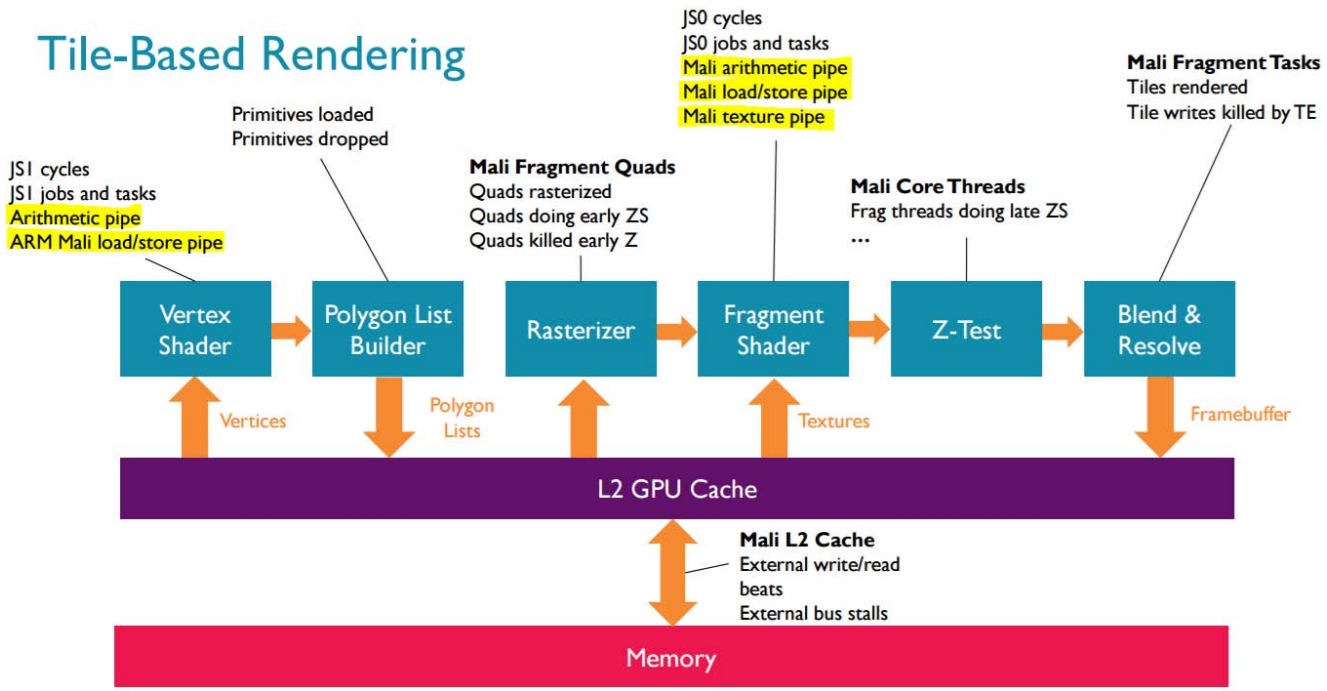
U.S. Patent No. 7,633,506: Claim 1

"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"

	 *See* https://community.arm.com/groups/arm-mali-graphics/blog/2012/08/17/how-low-can-you-go-building-low-power-low-bandwidth-arm-mali-gpus.

U.S. Patent No. 7,633,506: Claim 1

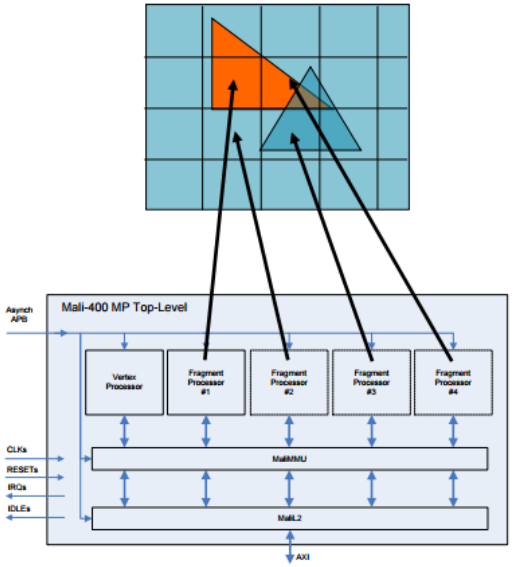
"a back-end in the graphics chip configured to receive said geometry and to process said geometry into one or more final pixels to be placed in a frame buffer;"



See ARM, How to Optimize Your Mobile Game with ARM Tools and Practical Examples, p.33, http://malideveloper.arm.com/downloads/GDC15/How%20to%20Optimize%20Your%20Mobile%20Game%20with%20ARM%20Tools%20and%20Practical%20Examples.pdf.

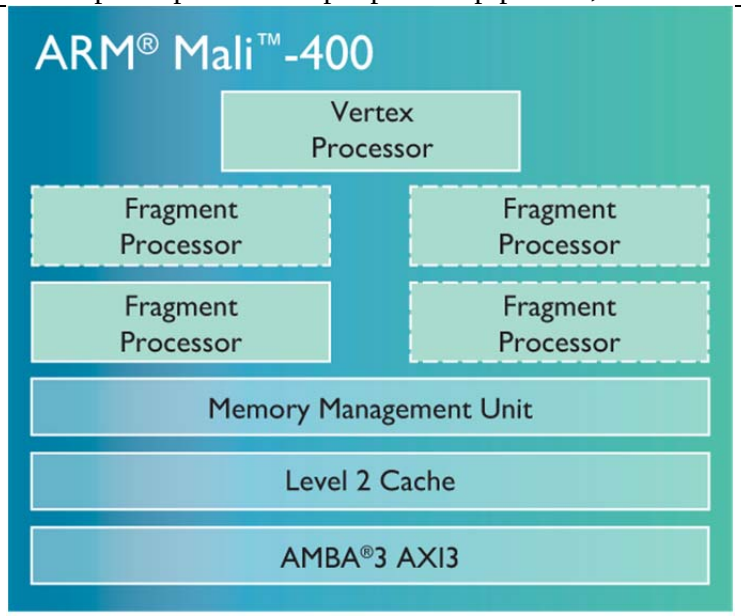
U.S. Patent No. 7,633,506: Claim 1

"wherein said back-end in the graphics chip comprises multiple parallel pipelines;"

wherein said back-end in the graphics chip comprises multiple parallel pipelines;	The VIZIO Product includes a back-end in the graphics chip that comprises multiple parallel pipelines. For example, each Fragment Processor "work[s] in parallel on separate tasks" and "processes one tile at a time until completion[.]" **Going Multi-Core** ▪ All cores work in parallel on separate tasks ▪ Each core processes one tile at a time until completion – no communication between cores ▪ Tiles assigned statically to cores in a swizzled order ▪ Tile processing order maximizes L2 hit rate for polygon descriptors, textures  mali Bringing Visual Entertainment to Life 12 ARM *See* Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.12.

U.S. Patent No. 7,633,506: Claim 1

"wherein said back-end in the graphics chip comprises multiple parallel pipelines;"



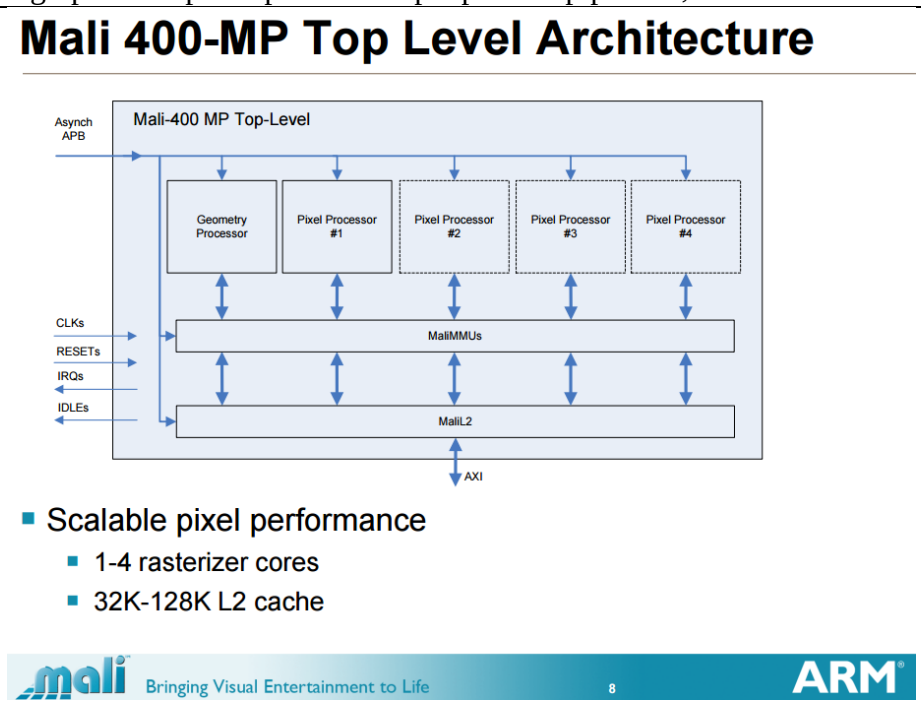
See https://www.arm.com/products/multimedia/mali-gpu/ultra-low-power/mali-400.php.

> Mali-400 MP is a multicore GPU: it contains a vertex processor and up to four fragment processors. The vertex processor is a core and the fragment processors are also cores: they run independently on separate tasks and contain their own critical resources. Within each Mali-400 fragment processor there is one pipeline, however it has a very complex pipeline which has several sub-pipelines to handle different tasks.

See https://community.arm.com/groups/arm-mali-graphics/blog/2011/03/28/multicore-or-multi-pipe-gpus-easy-steps-to-becoming-multi-frag-gasmic.

U.S. Patent No. 7,633,506: Claim 1

"wherein said back-end in the graphics chip comprises multiple parallel pipelines;"



See Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.8.

U.S. Patent No. 7,633,506: Claim 1

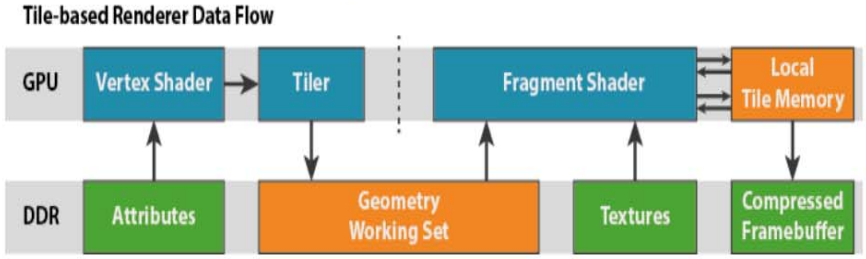
"wherein said back-end in the graphics chip comprises multiple parallel pipelines;"

The Mali Approach

The Mali GPU family takes a very different approach, commonly called tile-based rendering, designed to minimize the amount of power hungry external memory accesses which are needed during rendering. As described in the first blog in this series, Mali uses a distinct two-pass rendering algorithm for each render target. It first executes all of the geometry processing, and then executes all of the fragment processing. During the geometry processing stage, Mali GPUs break up the screen into small 16x16 pixel tiles and construct a list of which rendering primitives are present in each tile. When the GPU fragment shading step runs, each shader core processes one 16x16 pixel tile at a time, rendering it to completion before starting the next one. For tile-based architectures the algorithm equates to:

```
01.    foreach( tile )
02.         foreach( primitive in tile )
03.              foreach( fragment in primitive in tile )
04.                   render fragment
05.
```

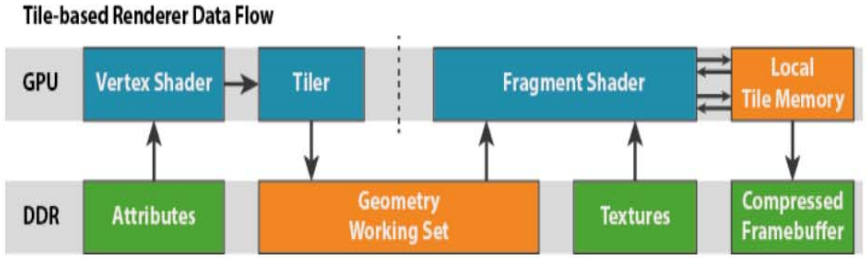
As a 16x16 tile is only a small fraction of the total screen area it is possible to keep the entire working set (color, depth, and stencil) for a whole tile in a fast RAM which is tightly coupled with the GPU shader core.



See https://community.arm.com/groups/arm-mali-graphics/blog/2014/02/20/the-mali-gpu-an-abstract-machine-part-2.

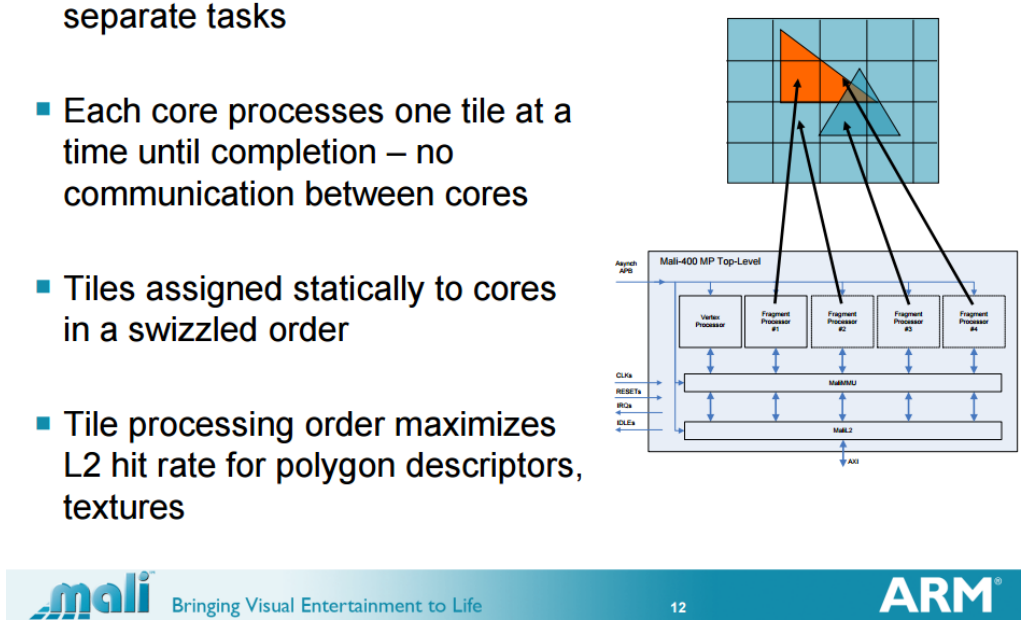
U.S. Patent No. 7,633,506: Claim 1

"wherein said geometry is determined to locate in a portion of an output screen defined by a tile; and"

wherein said geometry is determined to locate in a portion of an output screen defined by a tile; and	The geometry is determined to locate in a portion of an output screen defined by a tile.
	
	For example, the Mali GPUs implement tiled-based rendering.
	
	The Mali Approach
	
	The Mali GPU family takes a very different approach, commonly called tile-based rendering, designed to minimize the amount of power hungry external memory accesses which are needed during rendering. As described in the first blog in this series, Mali uses a distinct two-pass rendering algorithm for each render target. It first executes all of the geometry processing, and then executes all of the fragment processing. During the geometry processing stage, Mali GPUs break up the screen into small 16x16 pixel tiles and construct a list of which rendering primitives are present in each tile. When the GPU fragment shading step runs, each shader core processes one 16x16 pixel tile at a time, rendering it to completion before starting the next one. For tile-based architectures the algorithm equates to:
	
	`foreach( tile )`
	`  foreach( primitive in tile )`
	`  foreach( fragment in primitive in tile )`
	`  render fragment`
	
	As a 16x16 tile is only a small fraction of the total screen area it is possible to keep the entire working set (color, depth, and stencil) for a whole tile in a fast RAM which is tightly coupled with the GPU shader core.
	
	 
	
	See https://community.arm.com/groups/arm-mali-graphics/blog/2014/02/20/the-mali-gpu-an-abstract-machine-part-2.
	
	Furthermore, as depicted below, the geometry is determined to locate in a portion of an output screen defined by a tile.

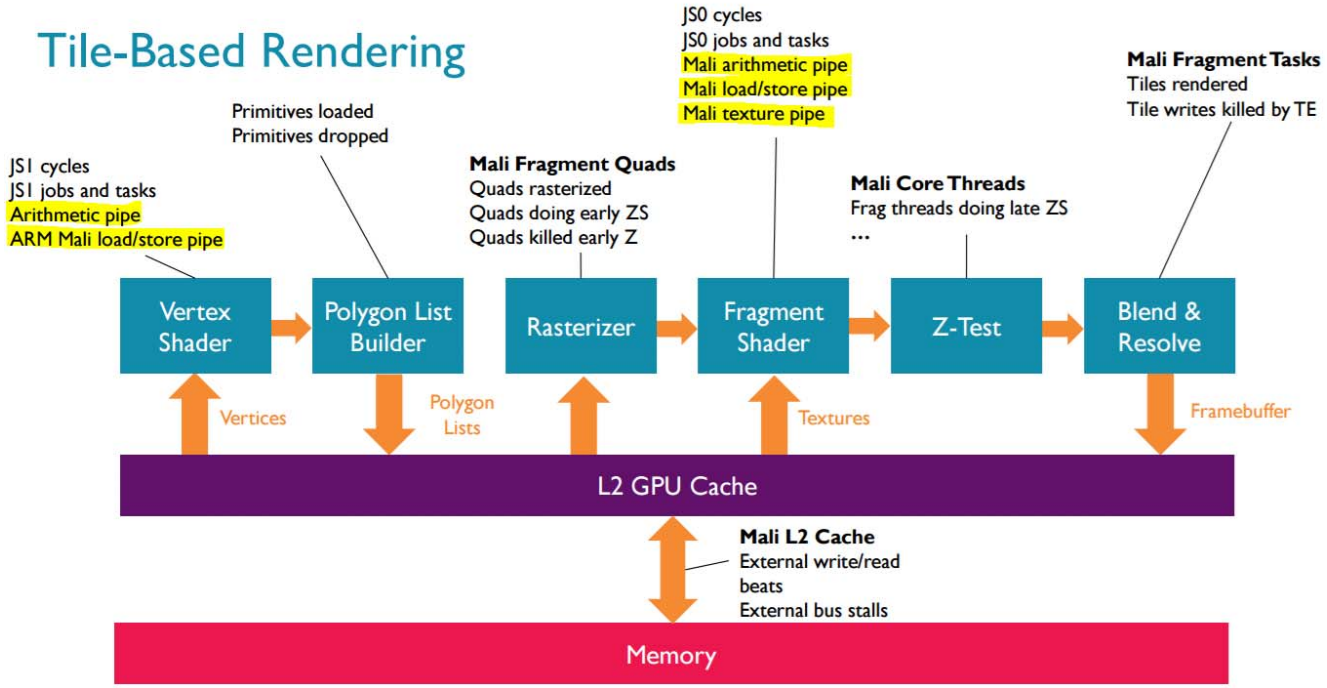
U.S. Patent No. 7,633,506: Claim 1

"wherein said geometry is determined to locate in a portion of an output screen defined by a tile; and"



See Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.12.

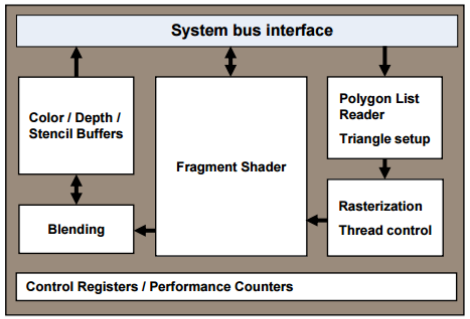
"wherein said geometry is determined to locate in a portion of an output screen defined by a tile; and"



See ARM, How to Optimize Your Mobile Game with ARM Tools and Practical Examples, p.33, http://malideveloper.arm.com/downloads/GDC15/How%20to%20Optimize%20Your%20Mobile%20Game%20with%20ARM%20Tools%20and%20Practical%20Examples.pdf.

U.S. Patent No. 7,633,506: Claim 1

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading.	Each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading. For example, each Fragment Processor includes a fragment shader. **Mali-400 MP Pixel Processor** ▪ Fragment Shader ▪ 128-thread barrel processor ▪ Fully general control flow ▪ VLIW ISA, tuned for graphics ▪ One texture sample per clock ▪ Key Features ▪ 16x16 on-chip tile buffer ▪ Renders one pixel per clock: 275M pix/sec @ 275 MHz ▪ No penalty for 4x MSAA ▪ No penalty for blending ▪ 4x or 16x MSAA resolve on output ▪ OpenGL ES 2.0 states handled without state dependent shaders   *See* Mali-400 MP: A Scalable GPU for Mobile Devices, available at http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_ARM.pdf, at p.11. Moreover, “the fragment shader receives interpolated texture coordinates from the vertex shader and performs a texture lookup to obtain the color at these coordinates…[i]t combines this color with the ambient color to produce the final output color.”

"wherein each of said parallel pipelines further comprises a unified shader that is programmable to perform both color shading and texture shading."

	6.1.9 Fragment Shaders The fragment shader is the graphics pipeline stage after primitive rasterization. For each sample of the pixels covered by a primitive, a fragment is generated. The fragment shader code is executed for each generated fragment. There are many more fragments than vertices so you must take care about the number of operations performed in the fragment shader. In the fragment shader you can access the fragment coordinates in the windows space among other values that contains all interpolated per-vertex output values from the vertex shader. In the shader example in *6.1.2 Shader structure* on page 6-72, the fragment shader receives the interpolated texture coordinates from the vertex shader and performs a texture lookup to obtain the color at these coordinates. It combines this color with the ambient color to produce the final output color. From the declaration of the fragment shader `float4 frag(vertexOutput input) : COLOR` it is clear that it is expected to produce the fragment color. The fragment shader is where you do the operations to achieve the required effect. This ultimately consists of assigning the correct color to a fragment. *See* "ARM Guide to Unity" Version 2.1, available at http://infocenter.arm.com/help/topic/com.arm.doc.100140_0201_00_en/arm_guide_to_unity_enhancing_your_mobile_games_100140_0201_00_en.pdf, at p. 6-79.