Metal feature set tables

Developer

This table lists each current Apple GPU family, its processors, and how each family relates to older feature sets.

Apple GPUs

Apple GPU family[1]	GPUs in family	Corresponding feature sets
Apple2	A8	iOS GPU Family 2 tvOS GPU Family 1
Apple3	A9, A10	iOS GPU Family 3 tvOS GPU Family 2
Apple4	A11	iOS GPU Family 4
Apple5	A12	iOS GPU Family 5
Apple6	A13	—
Apple7	A14 M1, M1 Pro, M1 Max, M1 Ultra	—
Apple8	A15, A16 M2, M2 Pro, M2 Max	—
Apple9	A17 Pro M3, M3 Pro, M3 Max	—

1. See `MTLGPUFamily` for each GPU family's enumeration constant.

When an Apple GPU is installed in a Mac device (Apple Silicon Mac), the device also reports support for the `mac2` GPU family; these devices support the union of both feature families.

This table lists each current Metal 3 GPU family and the processors in that family.

Metal 3 GPUs

Metal GPU family[1]	Platform	GPUs in family
Metal3	iOS	A14, A15, A16, A17 Pro
	iPadOS	A14, A15, A16 M1, M2
	macOS	M1, M1 Pro, M1 Max, M1 Ultra M2, M2 Pro, M2 Max M3, M3 Pro, M3 Max AMD Vega AMD 5000-series, 6000-series Intel UHD Graphics 630 Intel Iris Plus Graphics

1. See `MTLGPUFamily` for each GPU family's enumeration constant.

Metal feature availability by GPU family

GPU family[1]	Common1	Common2	Common3	Metal3	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
Feature	**Available in family**												
MetalKit	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Metal Performance Shaders		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Programmable blending					✓	✓	✓	✓	✓	✓	✓	✓	
PVRTC pixel formats					✓	✓	✓	✓	✓	✓	✓	✓	
EAC/ETC pixel formats					✓	✓	✓	✓	✓	✓	✓	✓	
ASTC pixel formats					✓	✓	✓	✓	✓	✓	✓	✓	
BC pixel formats[2]				Varies						Varies	Varies	✓	✓
Compressed volume texture formats		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Extended range pixel formats						✓	✓	✓	✓	✓	✓	✓	
Wide color pixel format	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Depth-16 pixel format	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linear textures	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSAA depth resolve			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Array of textures (read)		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Array of textures (write)		✓	✓	✓					✓	✓	✓	✓	✓
Cube map texture arrays		✓	✓	✓			✓	✓	✓	✓	✓	✓	✓
Stencil texture views	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Array of samplers		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Sampler max anisotropy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sampler LOD clamp	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MTLSamplerState support for comparison functions		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
16-bit unsigned integer coordinates	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Border color				✓						✓	✓	✓	✓
Counting occlusion query		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Base vertex/instance drawing		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Layered rendering			✓	✓				✓	✓	✓	✓	✓	✓
Layered rendering to multisample textures			✓	✓						✓	✓	✓	✓
Memoryless render targets					✓	✓	✓	✓	✓	✓	✓	✓	
Dual-source blending	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Combined MSAA store and resolve action		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
MSAA blits	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Programmable sample positions	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Deferred store action	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Texture barriers													✓
Memory barriers[3]		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Memory barriers in indirect command buffers (compute)		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Memory barriers in indirect command buffers (rendering)												✓	

GPU family[1]	Common1	Common2	Common3	Metal3	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
Tessellation		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Indirect tessellation arguments				✓				✓	✓	✓	✓	✓	✓
Tessellation in indirect command buffers				✓				✓	✓	✓	✓	✓	✓
Resource heaps	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Function specialization	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Read/write buffers in functions		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Read/write textures in functions			✓	✓			✓	✓	✓	✓	✓	✓	✓
Extract, insert, and reverse bits	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SIMD barrier	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Indirect draw & dispatch arguments		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Argument buffers tier	Varies	Varies	Varies	Tier 2	Tier 1	Tier 1	Tier 1	Tier 1	Tier 2	Tier 2	Tier 2	Tier 2	Tier 2
Indirect command buffers (rendering)		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Indirect command buffers (compute)		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Uniform type	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Imageblocks							✓	✓	✓	✓	✓	✓	
Tile shaders							✓	✓	✓	✓	✓	✓	
Imageblock sample coverage control							✓	✓	✓	✓	✓	✓	
Post-depth coverage							✓	✓	✓	✓	✓	✓	
Quad-scoped permute operations			✓	✓			✓	✓	✓	✓	✓	✓	✓
SIMD-scoped permute operations				✓					✓	✓	✓	✓	✓
SIMD-scoped reduction operations				✓						✓	✓	✓	✓
SIMD-scoped matrix multiply operations										✓	✓	✓	
Raster order groups[4]			✓	✓			✓	✓	✓	✓	✓	✓	Varies
Non-uniform threadgroup size			✓	✓			✓	✓	✓	✓	✓	✓	✓
Multiple viewports			✓	✓				✓	✓	✓	✓	✓	✓
Device notifications													✓
Stencil feedback			✓	✓				✓	✓	✓	✓	✓	✓
Stencil resolve			✓	✓				✓	✓	✓	✓	✓	✓
Non-square tile dispatch								✓	✓	✓	✓	✓	
Texture swizzle				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Placement heap				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Primitive ID				✓						✓	✓	✓	✓
Barycentric coordinates[5]				Varies						✓	✓	✓	Varies
Read/write cube map textures in functions				✓			✓	✓	✓	✓	✓	✓	✓
Sparse textures									✓	✓	✓	✓	
Sparse depth and stencil textures											✓	✓	
Variable rasterization rate[6]									✓	✓	✓	✓	Varies
Vertex amplification[7]									✓	✓	✓	✓	Varies
64-bit integer math				✓		✓	✓	✓	✓	✓	✓	✓	

GPU family[1]	Common1	Common2	Common3	Metal3	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
Lossy texture compression											✓	✓	
SIMD shift and fill											✓	✓	
Render dynamic libraries									✓	✓	✓	✓	
Compute dynamic libraries				✓					✓	✓	✓	✓	✓
Mesh shading				✓						✓	✓	✓	✓
MetalFX spatial upscaling[8]		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
MetalFX temporal upscaling[9]				Varies						✓	✓	✓	
Fast resource loading	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Ray tracing in compute pipelines[10]				✓					✓	✓	✓	✓	Varies
Ray tracing in render pipelines[11]									✓	✓	✓	✓	
Floating point atomics										✓	✓	✓	✓
Texture atomics				✓					✓	✓	✓	✓	✓
64-bit atomics[12]											Varies	✓	
Query texture LOD[13]										Varies	✓	✓	
Binary archives		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
Function pointers in compute pipelines[14]				✓					✓	✓	✓	✓	Varies
Function pointers in render pipelines[11]									✓	✓	✓	✓	
Depth sample compare bias and gradient					✓	✓	✓	✓	✓	✓	✓	✓	
Non-private depth stencil textures					✓	✓	✓	✓	✓	✓	✓	✓	
Dynamic stride for attribute buffers			✓	✓			✓	✓	✓	✓	✓	✓	✓
`MTLAttributeFormat`.`floatRGB9E5` and .`floatRG11B10`			✓	✓				✓	✓	✓	✓	✓	✓
`MTLDataType`.`bfloat` (brain float) scalar and vector cases				✓					✓	✓	✓	✓	✓

1. See `MTLGPUFamily` for each GPU family's enumeration constant.
2. Some GPU devices in the Apple7 and Apple8 families support BC texture compression on iPadOS. You can check whether a GPU supports BC texture compression by inspecting its `MTLDevice.supportsBCTextureCompression` property at runtime.
3. GPU devices in Apple3 through Apple9 families don't support memory barriers that include the `MTLRenderStages.fragment` or `.tile` stages in the `after` argument, or `MTLBarrierScope.renderTargets` in the `scope` argument of `MTLRenderCommandEncoder.memoryBarrier(scope:after:before:)` and `MTLRenderCommandEncoder.memoryBarrier(resources:after:before:)`.
4. Some GPU devices in the Mac2 family support raster order groups. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.rasterOrderGroupsSupported` property at runtime.
5. Some GPU devices in the Mac2 and Metal3 families support barycentric coordinates. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsShaderBarycentricCoordinates` property at runtime.
6. Some GPU devices in the Mac2 family support variable rasterization rates. You can check an individual GPU's support for this feature by calling its `MTLDevice.supportsRasterizationRateMap(layerCount:)` method at runtime.
7. Some GPU devices in the Mac2 family support vertex amplification. You can check an individual GPU's support for this feature by calling its `MTLDevice.supportsVertexAmplificationCount(_:)` method at runtime.
8. Apple TV devices don't support MetalFX. You can check whether a GPU supports spatial upscaling by calling the `MTLFXSpatialScalerDescriptor` type's `supportsDevice(_:)` method at runtime.
9. Apple TV devices don't support MetalFX. You can check whether a GPU supports temporal upscaling by calling the `MTLFXTemporalScalerDescriptor` type's `supportsDevice(_:)` method at runtime.
10. Some GPU devices in the Mac2 family support ray tracing in compute pipelines. You can check whether a GPU supports ray tracing in compute pipelines by inspecting its `MTLDevice.supportsRaytracing` property at runtime.
11. Support for function pointers and ray tracing in render pipelines isn't compatible with mesh shading; you can only use AIR linking through `MTLLinkedFunctions.privateFunctions` in render pipelines using mesh shading.
12. Some GPU devices in the Apple8 family support 64-bit atomic min and max using `ulong`, on both buffers and textures. You can check whether a GPU supports 64-bit atomics by verifying it supports both the Mac2 and Apple8 families.
13. Some GPU devices in the Apple7 family support query texture LOD. You can check whether a GPU supports query texture LOD by inspecting its `MTLDevice.supportsQueryTextureLOD` property at runtime.
14. Some GPU devices in the Mac2 family support function pointers in compute pipelines. You can check whether a GPU supports function pointers in compute pipelines by inspecting its `MTLDevice.supportsFunctionPointers` property at runtime.

GPU implementation limits by family

GPU family[1]	`Apple2`	`Apple3`	`Apple4`	`Apple5`	`Apple6`	`Apple7`	`Apple8`	`Apple9`	`Mac2`
Function arguments	**Function arguments**								
Maximum number of vertex attributes, per vertex descriptor	31	31	31	31	31	31	31	31	31
Maximum number of entries in the buffer argument table, per graphics or kernel function	31	31	31	31	31	31	31	31	31
Maximum number of entries in the texture argument table, per graphics or kernel function	31	31	96	96	128	128	128	128	128
Maximum number of entries in the sampler state argument table, per graphics or kernel function[2]	16	16	16	16	16	16	16	16	16
Maximum number of entries in the threadgroup memory argument table, per kernel function	31	31	31	31	31	31	31	31	31
Maximum number of constant buffer arguments in vertex, fragment, tile, or kernel function	31	31	31	31	31	31	31	31	14
Maximum length of constant buffer arguments in vertex, fragment, tile, or kernel function	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB
Maximum threads per threadgroup[3]	512	512	1024	1024	1024	1024	1024	1024	1024
Maximum total threadgroup memory allocation	16352 B	16 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB
Maximum total tile memory allocation[4]	Not accessible	Not accessible	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	Not accessible
Threadgroup memory length alignment	16 B	16 B	16 B	16 B	16 B	16 B	16 B	16 B	16 B
Maximum function memory allocation for a buffer in the constant address space	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit
Maximum scalars or vectors inputs to a fragment function. (Declare with the `[[stage_in]]` qualifier[4].)	60	60	124	124	124	124	124	124	32
Maximum number of input components to a fragment function. (Declare with the `[[stage_in]]` qualifier[5].)	60	60	124	124	124	124	124	124	124
Maximum number of function constants	65536	65536	65536	65536	65536	65536	65536	65536	65536
Maximum tessellation factor	Not available	16	16	64	64	64	64	64	64
Maximum number of viewports and scissor rectangles, per vertex function	1	1	1	16	16	16	16	16	16
Maximum number of raster order groups, per fragment function	Not available	Not available	8	8	8	8	8	8	8

GPU family[1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
Minimum alignment of buffer layout descriptor stride	4 B	4 B	4 B	1 B	1 B	1 B	1 B	1 B	4 B
Maximum size of buffer layout descriptor stride	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit	4 KB
Argument buffers[6]	**Argument buffers[6]**								
Maximum number of buffers you can access, per stage, from an argument buffer	31	31	96	96	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited
Maximum number of textures you can access, per stage, from an argument buffer	31	31	96	96	1 M	1 M	1 M	1 M	1 M
Maximum number of samplers you can access, per stage, from an argument buffer	16	16	16	16	1024	1024	1024	500 K	1024
Resources	**Resources**								
Minimum constant buffer offset alignment	4 B	4 B	4 B	4 B	4 B	4 B	4 B	4 B	32 B
Maximum 1D texture width	8192 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px
Maximum 2D texture width and height	8192 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px
Maximum cube map texture width and height	8192 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px	16384 px
Maximum 3D texture width, height, and depth	2048 px	2048 px	2048 px	2048 px	2048 px	2048 px	2048 px	2048 px	2048 px
Maximum texture buffer width[7]	64M px	256M px	256M px	256M px	256M px	256M px	256M px	256M px	256M px
Maximum number of layers per 1D texture array, 2D texture array, or 3D texture	2048	2048	2048	2048	2048	2048	2048	2048	2048
Buffer alignment for copying an existing texture to a buffer	64 B	16 B	16 B	16 B	16 B	16 B	16 B	16 B	256 B
Maximum counter sample buffer length	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	No limit
Maximum number of sample buffers	32	32	32	32	32	32	32	32	No limit
Render targets	**Render targets**								
Maximum number of color render targets per render pass descriptor	8	8	8	8	8	8	8	8	8
Maximum size of a point primitive	511	511	511	511	511	511	511	511	511

GPU family[1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
Maximum total render target size, per pixel, when using multiple color render targets	256 bits	256 bits	512 bits	512 bits	512 bits	512 bits	512 bits	512 bits	No limit
Maximum visibility query offset	65528 B	65528 B	65528 B	65528 B	65528 B	256 KB	256 KB	256 KB	256 KB
Feature limits	**Feature limits**								
Maximum number of fences	32768	32768	32768	32768	32768	32768	32768	32768	32768
Maximum number of I/O commands per buffer	8192	8192	8192	8192	8192	8192	8192	8192	8192
Maximum vertex count for vertex amplification[8]	Not available	Not available	Not available	Not available	2	8	8	8	Varies
Maximum threadgroups per object shader grid	Not available	Not available	Not available	Not available	Not available	No limit	No limit	No limit	1024
Maximum threadgroups per mesh shader grid[9]	Not available	Not available	Not available	Not available	Not available	1024	1024	1048575	1024
Maximum payload in mesh shader pipeline[10]	Not available	Not available	Not available	Not available	Not available	16384 B	16384 B	16384 B	16384 B
Largest number of levels a ray-tracing intersector can traverse in an acceleration structure[11]	Not available	Not available	Not available	Not available	32	32	32	32	32
Largest number of levels a ray-tracing intersection query can traverse in an acceleration structure[11]	Not available	Not available	Not available	Not available	16	16	16	16	16

1. See `MTLGPUFamily` for each GPU family's enumeration constant.
2. Inline `constexpr` samplers that you declare in Metal Shading Language (MSL) code count against the limit. For example, for a feature set limit of 16, you can have 12 API samplers and 4 language samplers (16 total), but you can't have 12 API samplers and 6 language samplers (18 total).
3. The values in this row are the theoretical maximum number of threads per threadgroup. Check the actual maximum by inspecting the `MTLComputePipelineState.maxTotalThreadsPerThreadgroup` property at runtime.
4. You can allocate memory between `imageblock` and `threadgroup` memory, but the sum of these allocations can't exceed the maximum total tile memory limit. Some feature sets can't access tile memory directly, but they can access threadgroup memory.
5. A vector counts as *n* scalars, where *n* is the number of components in the vector. The iOS and tvOS feature sets only reach the maximum number of inputs if you don't exceed the maximum number of input components. For example, you can have 60 float inputs (components), but you can't have 60 `float4` inputs, which total 240 components.
6. The limits apply to the items you place both in the argument buffers you bind directly and in the argument buffers you can access indirectly through your bound argument buffers.
7. The maximum texture buffer width, in pixels, is also limited by `MTLDevice.maxBufferLength` divided by the size of a pixel, in bytes; as well as available memory.
8. Some GPU devices in the Mac2 family support vertex amplification. You can check an individual GPU's support for a specific vertex amplification count by calling its `MTLDevice.supportsVertexAmplificationCount(_:)` method at runtime.
9. Mesh shaders can use up to 4 GB of payload and mesh geometry per draw for devices in the Apple7 and Apple8 GPU families.
10. Mesh shaders that have a `[[threadgroups_per_grid]]` or `[[threads_per_grid]]` parameter reduce the available payload size by 16 bytes. Viewing a mesh shader's geometry in the Metal debugger (within Xcode) reduces the available payload by 16 bytes. The total payload size reduction can be 32 bytes.
11. The value includes one level for the primitive acceleration structure, which leaves the remaining levels for instance acceleration structures.

This table lists the GPU's texture capabilities for each pixel format:

- **Atomic**: The GPU can use atomic operations on textures with the pixel format.
- **All**: The GPU has all of the texture capabilities below for the pixel format.
- **Filter**: The GPU can filter a texture with the pixel format during sampling.
- **Write**: The GPU can write to a texture on a per-pixel basis with the pixel format.[1]
- **Color**: The GPU can use a texture with the pixel format as a color render target.
- **Blend**: The GPU can blend a texture with the pixel format.
- **MSAA**: The GPU can use a texture with the pixel format as a destination for multisample antialias (MSAA) data.
- **Sparse**: The GPU supports sparse-texture allocations for textures with the pixel format.
- **Resolve**: The GPU can use a texture with the pixel format as a source for multisample antialias (MSAA) resolve operations.

Note
All graphics and compute kernels can read or sample a texture with any pixel format.

Texture capabilities by pixel format

GPU family[1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
Ordinary 8-bit pixel formats	Texture capabilities for **ordinary 8-bit pixel formats** by GPU Family								
A8Unorm[2,9]	Filter	All	All	All	All	All	All	All	All
R8Unorm[2]	All	All	All	All	All	All	All	All	All
R8Unorm_sRGB	All	All	All	All	All	All	All	All	Not available
R8Snorm	All	All	All	All	All	All	All	All	All
R8Uint[2] R8Sint[2]	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA
Ordinary 16-bit pixel formats	Texture capabilities for **ordinary 16-bit pixel formats** by GPU family								
R16Unorm R16Snorm	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	All
R16Uint[2] R16Sint[2]	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA
R16Float[2]	All	All	All	All	All	All	All	All	All

GPU family[1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
RG8Unorm	All	All	All	All	All	All	All	All	All
RG8Unorm_sRGB	All	All	All	All	All	All	All	All	Not available
RG8Snorm	All	All	All	All	All	All	All	All	All
RG8Uint RG8Sint	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA
Packed 16-bit pixel formats[7]	Texture capabilities for **packed 16-bit pixel formats** by GPU family								
B5G6R5Unorm A1BGR5Unorm ABGR4Unorm BGR5A1Unorm	Filter Color MSAA Resolve Blend	Filter Color MSAA Resolve Blend	Filter Color MSAA Resolve Blend	Filter Color MSAA Resolve Blend	Filter Color MSAA Resolve Blend Sparse	Filter Color MSAA Resolve Blend Sparse	Filter Color MSAA Resolve Blend Sparse	Filter Color MSAA Resolve Blend Sparse	Not available
Ordinary 32-bit pixel formats	Texture capabilities for **ordinary 32-bit pixel formats** by GPU family								
R32Uint[2] R32Sint[2]	Write Color	Write Color	Write Color	Write Color	Write Color Sparse Atomic	Write Color Sparse Atomic	Write Color Sparse Atomic	Write Color Sparse Atomic	Write Color MSAA Atomic
R32Float[2,6]	Write Color MSAA Blend	Write Color MSAA Blend	Write Color MSAA Blend	Write Color MSAA Blend	Write Color MSAA Blend Sparse	Write Color MSAA Blend Sparse	Write Color MSAA Blend Sparse	All	All
RG16Unorm RG16Snorm	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	All
RG16Uint RG16Sint	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA
RG16Float	All	All	All	All	All	All	All	All	All

GPU family[1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
RGBA8Unorm[2]	All	All	All	All	All	All	All	All	All
RGBA8Unorm_sRGB	All	All	All	All	All	All	All	All	Filter Color MSAA Resolve Blend
RGBA8Snorm	All	All	All	All	All	All	All	All	All
RGBA8Uint[2] RGBA8Sint[2]	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA
BGRA8Unorm	All	All	All	All	All	All	All	All	All
BGRA8Unorm_sRGB	All	All	All	All	All	All	All	All	Filter Color MSAA Resolve Blend
Packed 32-bit pixel formats	Texture capabilities for **packed 32-bit pixel formats** by GPU family								
RGB10A2Unorm	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All
BGR10A2Unorm	All	All	All	All	All	All	All	All	All
RGB10A2Uint	Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA
RG11B10Float[7]	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All
RGB9E5Float[7]	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	Filter

GPU family[1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
Ordinary 64-bit pixel formats	Texture capabilities for **ordinary 64-bit pixel formats** by GPU family								
RG32Uint[10] RG32Sint	Write Color	Write Color	Write Color	Write Color	Write Color Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse Atomic	Write Color MSAA Sparse Atomic	Write Color MSAA
RG32Float[6]	Write Color Blend	Write Color Blend	Write Color Blend	Write Color Blend	Write Color Blend Sparse	Write Color MSAA Blend Sparse	Write Color MSAA Blend Sparse	All	All
RGBA16Unorm RGBA16Snorm	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	Filter Write Color MSAA Blend Sparse	All
RGBA16Uint[2] RGBA16Sint[2]	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA
RGBA16Float[2]	All	All	All	All	All	All	All	All	All
Ordinary 128-bit pixel formats	Texture capabilities for **ordinary 128-bit pixel formats** by GPU family								
RGBA32Uint[2] RGBA32Sint[2]	Write Color	Write Color	Write Color	Write Color	Write Color Sparse	Write Color Sparse	Write Color Sparse	Write Color Sparse	Write Color MSAA
RGBA32Float[2,6]	Write Color	Write Color	Write Color	Write Color	Write Color Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	All	All
Compressed pixel formats[7]	Texture capabilities for **compressed pixel formats** by GPU family								
PVRTC pixel formats[3]	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
EAC/ETC pixel formats	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
ASTC pixel formats	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
HDR ASTC pixel formats	Not available	Not available	Not available	Not available	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
BC pixel formats	Not available	Not available	Not available	Not available	Not available	Varies[8]	Varies[8]	Filter Sparse	Filter

GPU family[1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Mac2
YUV pixel formats[4, 7]	Texture capabilities for **YUV pixel formats** by GPU family								
GBGR422 BGRG422	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
Depth and stencil pixel formats[7]	Texture capabilities for **depth and stencil pixel formats** by GPU family								
Depth16Unorm	Filter MSAA	Filter MSAA Resolve	Filter MSAA Resolve	Filter MSAA Resolve	Filter MSAA Resolve	Filter MSAA Resolve	Filter MSAA Resolve Sparse	Filter MSAA Resolve Sparse	Filter MSAA Resolve
Depth32Float	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve Sparse	MSAA Resolve Sparse	Filter MSAA Resolve
Stencil8	MSAA	MSAA	MSAA	MSAA	MSAA	MSAA	MSAA Sparse	MSAA Sparse	MSAA
Depth24Unorm_Stencil8[5]	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Filter MSAA Resolve
Depth32Float_Stencil8	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	Filter MSAA Resolve
X24_Stencil8	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	MSAA
X32_Stencil8	MSAA	MSAA	MSAA	MSAA	MSAA	MSAA	MSAA	MSAA	MSAA
Extended range and wide color pixel formats	Texture capabilities for **extended range and wide color formats** by GPU family								
BGRA10_XR BGRA10_XR_sRGB BGR10_XR BGR10_XR_sRGB	Not available	All	All	All	All	All	All	All	Not available

1. See `MTLGPUFamily` for each GPU family's enumeration constant.
2. Some GPUs support read-write textures — where a kernel can both read from and write to a texture. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.readWriteTextureSupport` property at runtime.
3. Only the GPUs in Apple 3 and Apple 4 families support `MTLSamplerAddressMode.clampToZero` for the PVRTC pixel formats.
4. The GPUs in Apple 6 through Apple 9 families don't support sparse textures with YUV pixel formats.
5. Some GPUs support `MTLPixelFormat.depth24Unorm_stencil8`. You can check an individual GPU's support for this pixel format by inspecting its `MTLDevice.isDepth24Stencil8PixelFormatSupported` property at runtime.
6. Some GPUs in the Apple 7 and Apple 8 families additionally support Filter and Resolve (and therefore, All) 32-bit float capabilities. You can check whether a GPU supports 32-bit float Filter and Resolve by inspecting the `MTLDevice.supports32BitFloatFiltering` property at runtime.
7. Formats in this group aren't compatible with lossy texture compression through `MTLTextureDescriptor.compressionType`.
8. Some GPU devices in the Apple7 and Apple8 families support filtering and sparse BC compressed textures on iPadOS. You can check whether a GPU supports BC texture compression by inspecting its `MTLDevice.supportsBCTextureCompression` property at runtime.
9. The `A8Unorm` pixel format is incompatible with imageblocks with explicit layout. Use either an `R8Unorm` texture view, or imageblocks with implicit layout.
10. You can only apply the `RG32Uint` format to a `ulong` texture on a GPU that supports the "64-bit atomics" feature.

Texture buffer pixel formats

These tables list the pixel formats that texture buffers support and the GPU's read/write access to textures with those formats:

- **All**: The GPU can use all the accesses below for a texture buffer with the pixel format.
- **Read**: The GPU can use `read` access for a texture buffer with the pixel format.
- **Write**: The GPU can use `write` access for a texture buffer with the pixel format.
- **Read/Write**[1]: The GPU can use `read_write` access for a texture buffer with the pixel format.

Note
The GPU capabilities are generally the same across all hardware families, but some GPUs have additional options.[2]

Ordinary 8-bit pixel formats	
Format	**Access**
A8Unorm	All
R8Unorm	All
R8Snorm	Read Write
R8Uint R8Sint	All

Ordinary 16-bit pixel formats	
Format	**Access**
R16Unorm R16Snorm	Read Write
R16Uint R16Sint	All
R16Float	All
RG8Unorm	Read Write
RG8Snorm	Read Write
RG8Uint RG8Sint	Read Write

Ordinary 32-bit pixel formats	
Format	**Access**
R32Uint R32Sint	All[3]
R32Float	All
RG16Unorm RG16Snorm	Read Write
RG16Uint RG16Sint	Read Write
RG16Float	Read Write
RGBA8Unorm	All
RGBA8Snorm	Read Write
RGBA8Uint RGBA8Sint	All
BGRA8Unorm	Read

Packed 32-bit pixel formats	
Format	**Access**
RGB10A2Unorm	Read Write
RGB10A2Uint	Read Write
RG11B10Float	Read Write

Ordinary 64-bit pixel formats	
Format	**Access**
RG32Uint RG32Sint	Read Write
RG32Float	Read Write
RGBA16Unorm RGBA16Snorm	Read Write
RGBA16Uint RGBA16Sint	All
RGBA16Float	All

Ordinary 128-bit pixel formats	
Format	**Access**
RGBA32Uint RGBA32Sint	All
RGBA32Float	All

1. GPUs with the Tier 2 feature set support `read_write` access to textures. You can check an individual GPU's support for this pixel format by inspecting its `MTLDevice.readWriteTextureSupport` property at runtime.
2. Some devices support this pixel format. Check a device by inspecting its `MTLDevice.depth24Stencil8PixelFormatSupported` property at runtime.
3. GPUs that support texture atomics (see feature availability by GPU family) also support atomics in read/write texture buffers with this pixel format.