Metal Feature Set Tables

Developer

Metal GPUs

GPU	Metal version	Apple family [1]
A8-series	Metal	Apple2
A9-series	Metal	Apple3
A10-series	Metal	Apple3
A11 Bionic	Metal	Apple4
A12-series	Metal	Apple5
A13 Bionic	Metal	Apple6
A14 Bionic	Metal 3 & 4	Apple7
A15 Bionic	Metal 3 & 4	Apple8
A16 Bionic	Metal 3 & 4	Apple8
A17 Pro	Metal 3 & 4	Apple9
A18-series	Metal 3 & 4	Apple9
A19-series	Metal 3 & 4	Apple10
M1-series	Metal 3 & 4	Apple7
M2-series	Metal 3 & 4	Apple8
M3-series	Metal 3 & 4	Apple9
M4-series	Metal 3 & 4	Apple9
M5-series	Metal 3 & 4	Apple10

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

Metal feature availability by GPU family

Feature [1]	Metal	Apple
Feature	**Programming model family**	**Hardware family**
MetalKit	Metal 3 & 4	Apple2
Metal performance shaders	Metal 3 & 4	Apple2
Programmable blending	Metal 3 & 4	Apple2
Separate depth and stencil attachments	Metal 3 & 4	Apple2
PVRTC pixel formats	Metal 3 & 4	Apple2
EAC/ETC pixel formats	Metal 3 & 4	Apple2
ASTC pixel formats	Metal 3 & 4	Apple2
BC pixel formats [2]	Metal 3 & 4	Apple9
Compressed volume texture formats	Metal 3 & 4	Apple3
Extended range pixel formats	Metal 3 & 4	Apple3
Wide color pixel format	Metal 3 & 4	Apple2
Depth-16 pixel format	Metal 3 & 4	Apple2
Linear textures	Metal 3 & 4	Apple2
MSAA depth resolve	Metal 3 & 4	Apple3
Array of textures (read)	Metal 3 & 4	Apple3
Array of textures (write)	Metal 3 & 4	Apple6
Cube map texture arrays	Metal 3 & 4	Apple4
Stencil texture views	Metal 3 & 4	Apple2
Array of samplers	Metal 3 & 4	Apple3
Sampler maximum anisotropy	Metal 3 & 4	Apple2
Sampler LOD clamp	Metal 3 & 4	Apple2
MTLSamplerState support for comparison functions	Metal 3 & 4	Apple3
16-bit unsigned integer coordinates	Metal 3 & 4	Apple2
Border color	Metal 3 & 4	Apple7
Counting occlusion query	Metal 3 & 4	Apple3
Base vertex/instance drawing	Metal 3 & 4	Apple3
Layered rendering	Metal 3 & 4	Apple5
Layered rendering to multisample textures	Metal 3 & 4	Apple7
Memoryless render targets	Metal 3 & 4	Apple2
Dual-source blending	Metal 3 & 4	Apple2
Combined MSAA store and resolve action	Metal 3 & 4	Apple3
MSAA blits	Metal 3 & 4	Apple2
Programmable sample positions	Metal 3 & 4	Apple2
Deferred store action	Metal 3 & 4	Apple2
Memory barriers [3]	Metal 3 & 4	Apple3
Indirect command buffer support for memory barriers (compute)	Metal 3 & 4	Apple3
Indirect command buffer support for memory barriers (rendering)	Metal 3 & 4	Apple9
Tessellation	Metal 3	Apple3
Indirect tessellation arguments	Metal 3	Apple5
Tessellation in indirect command buffers	Metal 3	Apple5

Feature [1]	Metal	Apple
Resource heaps	Metal 3 & 4	Apple2
Function specialization	Metal 3 & 4	Apple2
Read/write buffers in functions	Metal 3 & 4	Apple3
Read/write textures in functions	Metal 3 & 4	Apple4
Extract, insert, and reverse bits	Metal 3 & 4	Apple2
SIMD barrier	Metal 3 & 4	Apple2
Indirect draw and dispatch arguments	Metal 3 & 4	Apple3
Argument buffers tier 1	Metal 3 & 4	Apple2
Argument buffers tier 2	Metal 3 & 4	Apple6
Indirect command buffers (rendering)	Metal 3 & 4	Apple3
Indirect command buffers (compute)	Metal 3 & 4	Apple3
Uniform type	Metal 3 & 4	Apple2
Imageblocks	Metal 3 & 4	Apple4
Tile shaders	Metal 3 & 4	Apple4
Imageblock sample coverage control	Metal 3 & 4	Apple4
Post-depth coverage	Metal 3 & 4	Apple4
Quad-scoped permute operations	Metal 3 & 4	Apple4
Quad-scoped ballot operation	Metal 3 & 4	Apple6
Quad-scoped reduction operations	Metal 3 & 4	Apple7
SIMD-scoped permute operations	Metal 3 & 4	Apple6
SIMD-scoped reduction operations	Metal 3 & 4	Apple7
SIMD-scoped matrix multiply operations	Metal 3 & 4	Apple7
Raster order groups	Metal 3 & 4	Apple4
Nonuniform threadgroup size	Metal 3 & 4	Apple4
Multiple viewports	Metal 3 & 4	Apple5
Stencil feedback	Metal 3 & 4	Apple5
Stencil resolve	Metal 3 & 4	Apple5
Nonsquare tile dispatch	Metal 3 & 4	Apple5
Texture swizzle	Metal 3 & 4	Apple2
Placement heap	Metal 3 & 4	Apple2
Primitive ID	Metal 3 & 4	Apple7
Barycentric coordinates	Metal 3 & 4	Apple7
Read/write cube map textures in functions	Metal 3 & 4	Apple4
Sparse textures [4]	Metal 3 & 4	Apple6
Sparse depth and stencil textures [5]	Metal 3 & 4	Apple8
Variable rasterization rate	Metal 3 & 4	Apple6
Vertex amplification	Metal 3 & 4	Apple6
64-bit integer math	Metal 3 & 4	Apple3
Lossy texture compression	Metal 3 & 4	Apple8
SIMD shift and fill	Metal 3 & 4	Apple8
Render dynamic libraries	Metal 3 & 4	Apple6
Compute dynamic libraries	Metal 3 & 4	Apple6

Feature [1]	Metal	Apple
Mesh shading	Metal 3 & 4	Apple7
Indirect mesh draw arguments	Metal 3 & 4	Apple9
Indirect command buffers containing mesh draws	Metal 3 & 4	Apple9
MetalFX spatial upscaling	Metal 3 & 4	Apple3
MetalFX temporal upscaling	Metal 3 & 4	Apple7
MetalFX frame interpolation	Metal 3 & 4	Apple5
MetalFX denoised upscaling	Metal 3 & 4	Apple9
Fast resource loading	Metal 3 & 4	Apple2
Ray tracing in compute pipelines	Metal 3 & 4	Apple6
Ray tracing in render pipelines [6]	Metal 3 & 4	Apple6
Floating-point atomics	Metal 3 & 4	Apple7
Texture atomics	Metal 3 & 4	Apple6
64-bit atomics [7]	Metal 3 & 4	Apple9
Query texture LOD [8]	Metal 3 & 4	Apple8
Binary archives	Metal 3 & 4	Apple3
Function pointers in compute pipelines	Metal 3 & 4	Apple6
Function pointers in render pipelines [6]	Metal 3 & 4	Apple6
Depth sample compare bias and gradient	Metal 3 & 4	Apple2
Nonprivate depth stencil textures	Metal 3 & 4	Apple2
Dynamic stride for attribute buffers	Metal 3 & 4	Apple4
`MTLAttributeFormat.floatRGB9E5` and `.floatRG11B10`	Metal 3 & 4	Apple5
`MTLDataType.bfloat` (brain float) scalar and vector cases	Metal 3 & 4	Apple6
Relaxed math	Metal 3 & 4	Apple4
Global built-ins and bindings	Metal 3 & 4	Apple6
Memory coherence for textures and buffers in shaders	Metal 3 & 4	Apple6
Per-pipeline shader validation	Metal 3 & 4	Apple6
Shader logging	Metal 3 & 4	Apple6
Residency sets	Metal 3 & 4	Apple6
Acceleration structures containing row-major matrices	Metal 3 & 4	Apple9
Ray tracing with per-component motion interpolation	Metal 3 & 4	Apple9
Direct access to on-chip ray-intersection result storage	Metal 3 & 4	Apple9
Fragment visibility count accumulation	Metal 3 & 4	Apple7
Argument tables	Metal 4	Apple7
Command allocators	Metal 4	Apple7
Decoupled command queues and command buffers	Metal 4	Apple7
Texture view pools	Metal 3 & 4	Apple7
Command barriers	Metal 4	Apple7
Placement sparse buffers [9]	Metal 4	Apple8
Placement sparse textures [9]	Metal 4	Apple8
Dedicated compilation contexts	Metal 4	Apple7
Pipeline dataset serialization	Metal 4	Apple7
Flexible render pipeline state	Metal 4	Apple7

Feature [1]	Metal	Apple
Color attachment mapping	Metal 3 & 4	Apple7
Machine learning encoding	Metal 4	Apple7
Tensors	Metal 4	Apple7
Performance counter heaps	Metal 4	Apple7
Address-driven acceleration structure builds	Metal 4	Apple9
Acceleration structure build options	Metal 3 & 4	Apple9
Intersection function buffers	Metal 3 & 4	Apple9
Sampler minimum and maximum reduction	Metal 3 & 4	Apple10
Sampler LOD bias	Metal 3 & 4	Apple10
Access to pre-raster per-vertex values	Metal 3 & 4	Apple10
Depth bounds testing	Metal 3 & 4	Apple10
Indirect command buffer support for raster and depth/stencil state	Metal 3 & 4	Apple10
Atomics on cube map and cube map array textures	Metal 3 & 4	Apple10
Universal texture compression	Metal 3 & 4	Apple10
Acquire and release memory ordering atomics	Metal 3 & 4	Apple7
2 x 1 and 1 x 2 block texture reads	Metal 3 & 4	Apple7
Clamp to edge behavior for texture reads	Metal 3 & 4	Apple7
Integer coordinate offsets for texture reads and atomics	Metal 3 & 4	Apple7

1. This table identifies feature availability across the Metal 3 and Metal 4 programming models. The Metal 3 programming model is the combined Metal 1, 2, and 3 API surface and is available on all Apple GPU families; the Metal 4 programming model is available as of `Apple7`. A feature is available when the programming model you are targeting appears in its Metal column, starting with GPU devices in the hardware family listed in its Apple column. See `MTLGPUFamily` for each GPU family's enumeration constant.
2. Some GPU devices in the `Apple7` and `Apple8` families support BC texture compression in iPadOS. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsBCTextureCompression` property at runtime. As of `Apple9` all GPUs have support.
3. GPU devices in `Apple3` through `Apple10` 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. GPU devices starting with the `Apple6` family support sparse color textures with automatic heap backing. Automatic heap backing doesn't support sparse buffers or sparse textures with `1D`, `1DArray`, or `TextureBuffer` texture types.
5. Some GPU devices in the `Apple7` family, including all `Apple7` macOS devices, support sparse depth and stencil textures with placement heap backing. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsPlacementSparse` property at runtime. As of `Apple8`, all GPUs have support for both placement and automatic heap backing for sparse color, depth, and stencil textures.
6. Support for function pointers and ray tracing in render pipelines isn't compatible with mesh shading. You can only use Metal IR linking through `MTLLinkedFunctions.privateFunctions` in render pipelines using mesh shading.
7. GPU devices in the `Apple8` family support 64-bit atomic minimum and maximum using `ulong`, on both buffers and textures, only on macOS. The full set of 64-bit atomic operations is supported on all platforms starting with `Apple9`.
8. Some GPU devices in the `Apple7` family support query texture LOD. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsQueryTextureLOD` property at runtime. As of `Apple8` all GPUs have support.
9. Some GPU devices in the `Apple7` family, including all `Apple7` macOS devices, support sparse buffers and textures with placement heap backing. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsPlacementSparse` property at runtime. As of `Apple8` all GPUs have support.

GPU implementation limits by family

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
Function arguments	**Function arguments**								
Maximum number of vertex attributes, per vertex descriptor [2]	31	31	31	31	31	31	31	31	31
Maximum number of entries in the buffer argument table, per graphics or kernel function [2]	31	31	31	31	31	31	31	31	31
Maximum number of entries in the texture argument table, per graphics or kernel function [2]	31	31	96	96	128	128	128	128	128
Maximum number of entries in the sampler state argument table, per graphics or kernel functions [2] [3]	16	16	16	16	16	16	16	16	16
Maximum number of entries in the threadgroup memory argument table, per kernel function [2]	31	31	31	31	31	31	31	31	31
Maximum number of constant buffer arguments in vertex, fragment, tile, or kernel functions [2]	31	31	31	31	31	31	31	31	31
Maximum length of inlined buffer contents using `setBytes` [4]	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB	4 KB
Maximum threads per threadgroup [5]	512	512	1024	1024	1024	1024	1024	1024	1024
Maximum total threadgroup memory allocation	16,352 B	16 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB
Maximum explicit image block allocation [6]	Not available	Not available	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB
Maximum implicit image block allocation [6]	Not available	Not available	128 KB	128 KB	128 KB	128 KB	128 KB	256 KB	256 KB
Threadgroup memory length alignment	16 B	16 B	16 B	16 B	16 B	16 B	16 B	16 B	16 B
Maximum scalar or vector inputs to a fragment function. (Declare with the `[[stage_in]]` qualifier.) [7]	60	60	124	124	124	124	124	124	124
Maximum number of input components to a fragment function. (Declare with the `[[stage_in]]` qualifier.) [7]	60	60	124	124	124	124	124	124	124
Maximum number of function constants	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536
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
Minimum alignment of buffer layout descriptor stride and attribute descriptor offset	4 B	4 B	4 B	1 B	1 B	1 B	1 B	1 B	1 B
Maximum size of buffer layout descriptor stride	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
Argument buffers [8]	**Argument buffers**								
Maximum number of buffers you can access, per stage, from an argument buffer	31	31	96	96	No limit	No limit	No limit	No limit	No limit
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	128	996	996	500 K	500 K
Resources	**Resources**								
Minimum constant buffer offset alignment	4 B	4 B	4 B	4 B	4 B	4 B	4 B	4 B	4 B
Maximum 1D texture width	8192 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	32,768 px
Maximum 2D texture width and height	8192 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	32,768 px
Maximum cube map texture width and height	8192 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	16,384 px	32,768 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 [9]	64 M px	256 M px	256 M px	256 M px	256 M px	256 M px	256 M px	256 M px	256 M px
Maximum number of layers per 1D texture array, 2D texture array, or 3D texture array	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	16 B
Maximum counter sample buffer length	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB
Maximum number of sample buffers	32	32	32	32	32	32	32	32	32
Maximum number of residency sets per queue	Not available	Not available	Not available	Not available	32	32	32	32	32
Maximum number of residency sets per buffer	Not available	Not available	Not available	Not available	32	32	32	32	32
Maximum indirect command buffer length	Not available	No limit	No limit	No limit	No limit	No limit	No limit	No limit	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
Maximum explicit image block size, per pixel, per sample, when using multiple color render targets	Not available	Not available	64 B	64 B	64 B	64 B	64 B	64 B	64 B

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
Maximum implicit image block size, per pixel, per sample, when using multiple color render targets	32 B	32 B	64 B	64 B	64 B	128 B	128 B	128 B	128 B
Maximum visibility query offset	65,528 B	65,528 B	65,528 B	65,528 B	65,528 B	256 KB	256 KB	256 KB	256 KB
Maximum sample count in render passes with MSAA	4	4	4	4	4	4	4	4	8
Maximum tile size in render passes without MSAA	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32
Maximum tile size in render passes with 2x MSAA	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32	32 x 32
Maximum tile size in render passes with 4x MSAA	32 x 16	32 x 16	32 x 16	32 x 16	32 x 16	32 x 16	32 x 16	32 x 16	32 x 16
Maximum tile size in render passes with 8x MSAA	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	16 x 16
Feature limits	**Feature limits**								
Maximum number of fences	32,768	32,768	32,768	32,768	32,768	32,768	32,768	32,768	32,768
Maximum number of I/O commands per buffer	8192	8192	8192	8192	8192	8192	8192	8192	8192
Maximum vertex count for vertex amplification	Not available	Not available	Not available	Not available	2	8	8	8	8
Maximum threadgroups per object shader grid	Not available	Not available	Not available	Not available	Not available	No limit	No limit	No limit	No limit
Maximum threadgroups per mesh shader grid [10]	Not available	Not available	Not available	Not available	Not available	1024	1024	1,048,575	4,194,303
Maximum payload in mesh shader pipeline [11]	Not available	Not available	Not available	Not available	Not available	16,384 B	16,384 B	16,384 B	16,384 B
Largest number of levels a ray-tracing intersector can traverse in an acceleration structure [12]	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 [12]	Not available	Not available	Not available	Not available	16	16	16	16	16
Maximum texture view pool entries	Not available	Not available	Not available	Not available	Not available	128 million	128 million	256 million	256 million
Maximum performance counter heaps (per process)	Not available	Not available	Not available	Not available	Not available	32	32	32	32
Minimum alignment of intersection function buffer	Not available	Not available	Not available	Not available	Not available	64 B	64 B	64 B	64 B
Minimum alignment of intersection function buffer stride	Not available	Not available	Not available	Not available	Not available	8 B	8 B	8 B	8 B

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
Maximum size of intersection function buffer stride	Not available	Not available	Not available	Not available	Not available	4096 B	4096 B	4096 B	4096 B

1. See `MTLGPUFamily` for each GPU family's enumeration constant. All Metal 3 and Metal 4 devices support at least the limits listed in the `Apple7` column.
2. These values are identical to the maximum number of bindings in an `MTL4ArgumentTable` of the same type.
3. Inline `constexpr` samplers that you declare in Metal Shading Language (MSL) code count toward 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).
4. Inlined buffer contents populate through functions like `setBytes`, and its variants for specific render stages. Noninlined buffer contents that you access through `MTLBuffer` or it's GPU virtual address are limited only by the size of that buffer.
5. 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.
6. You can allocate memory between `imageblock` and `threadgroup` memory, but the sum of these allocations can't exceed the maximum total image block memory limit. Some feature sets can't access image block memory directly, but they can access threadgroup memory. Which image block memory limit applies depends on the shader's usage of either implicit or explicit image block layout; see the Metal Shading Language specification for details.
7. 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.
8. The limits apply to the items you place in the argument buffers you bind directly and in the argument buffers you can access indirectly through your bound argument buffers.
9. 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.
10. Mesh shaders can use up to 4 GB of payload and mesh geometry per draw for devices in the `Apple7` and `Apple8` GPU families.
11. 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.
12. 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 the following texture capabilities 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.[2]
  - **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.
    Sparse isn't included in **All** for the `Apple2` through `Apple5` family columns, because those GPUs don't support the sparse texture feature.
  - **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	Apple10
Ordinary 8-bit pixel formats	Texture capabilities for **ordinary 8-bit pixel formats** by GPU family								
`A8Unorm` [2,3]	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	All
`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 Sparse
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	All	All	All	All	All	All	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 Sparse
`R16Float` [2]	All	All	All	All	All	All	All	All	All
`RG8Unorm`	All	All	All	All	All	All	All	All	All

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
RG8Unorm_sRGB	All	All	All	All	All	All	All	All	All
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 Sparse
Packed 16-bit pixel formats [4]	Texture capabilities for **packed 16-bit pixel formats** by GPU family								
B5G6R5Unorm A1BGR5Unorm [5] ABGR4Unorm [5] 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	Filter Color MSAA Resolve Blend Sparse
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	Atomic Write Color Sparse	Atomic Write Color MSAA Sparse	Atomic Write Color MSAA Sparse	Atomic Write Color MSAA Sparse	Atomic Write Color MSAA Sparse
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	All	All	All	All	All	All	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 Sparse
RG16Float	All	All	All	All	All	All	All	All	All
RGBA8Unorm [2]	All	All	All	All	All	All	All	All	All

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
`RGBA8Unorm_sRGB`	All	All	All	All	All	All	All	All	All
`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 Sparse
`BGRA8Unorm`	All	All	All	All	All	All	All	All	All
`BGRA8Unorm_sRGB`	All	All	All	All	All	All	All	All	All
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 Sparse
`RG11B10Float` [4]	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All
`RGB9E5Float` [4]	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All
Ordinary 64-bit pixel formats	Texture capabilities for **ordinary 64-bit pixel formats** by GPU family								

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
`RG32Uint` [7] `RG32Sint`	Write Color	Write Color	Write Color	Write Color	Write Color Sparse	Write Color MSAA Sparse	Atomic Write Color MSAA Sparse	Atomic Write Color MSAA Sparse	Atomic Write Color MSAA Sparse
`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	All	All	All	All	All	All	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 Sparse
`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 MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse	Write Color MSAA Sparse
`RGBA32Float` [2,6]	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
Compressed pixel formats [4]	Texture capabilities for **compressed pixel formats** by GPU family								
PVRTC pixel formats [8]	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse
EAC/ETC pixel formats	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse
ASTC pixel formats	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse
HDR ASTC pixel formats	Not available	Not available	Not available	Not available	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse
BC pixel formats	Not available	Not available	Not available	Not available	Not available	Varies [9]	Varies [9]	Filter Sparse	Filter Sparse

GPU family [1]	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10
YUV pixel formats [4]	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 [4]	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 Sparse [10]	Filter MSAA Resolve Sparse	Filter MSAA Resolve Sparse	Filter MSAA Resolve Sparse
`Depth32Float` [6]	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve Sparse [10]	MSAA Resolve Sparse	Filter MSAA Resolve Sparse	Filter MSAA Resolve Sparse
`Stencil8`	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve Sparse [10]	MSAA Resolve Sparse	MSAA Resolve Sparse	MSAA Resolve Sparse
`Depth32Float_Stencil8`	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	Filter MSAA Resolve	Filter MSAA Resolve
`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	All

1. See `MTLGPUFamily` for each GPU family's enumeration constant. All Metal 3 and Metal 4 devices support at least the texture capabilities listed in the `Apple7` column.
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. The `A8Unorm` pixel format is incompatible with imageblocks with explicit layout. Use either an `R8Unorm` texture view, or imageblocks with implicit layout.
4. Formats in this group aren't compatible with lossy texture compression through `MTLTextureDescriptor.compressionType`.
5. Textures with the `A1BGR5Unorm` or `ABGR4Unorm` format are incompatible with samplers using the `opaqueBlack` border color.
6. Some GPUs in the `Apple7` and `Apple8` families additionally support the **Filter** and **Resolve** texture capabilities for 32-bit floating-point pixel formats in iPadOS. You can check an individual GPU's support for this feature by inspecting the `MTLDevice.supports32BitFloatFiltering` property at runtime.
7. You can only apply the `RG32Uint` format to a `ulong` texture on a GPU that supports the 64-bit atomics feature.
8. Only the GPUs in `Apple3` and `Apple4` families support `MTLSamplerAddressMode.clampToZero` for the PVRTC pixel formats.
9. Some GPU devices in the `Apple7` and `Apple8` families support filtering and sparse BC-compressed textures in iPadOS. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsBCTextureCompression` property at runtime. All `Apple7` and `Apple8` macOS devices have support.
10. Some GPUs in the `Apple7` family in iPadOS, as well as all GPUs in the `Apple7` family in macOS, support **Sparse** depth and stencil textures with placement heap backing. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsPlacementSparse` property at runtime.

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 the following accesses 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**: The GPU can use `read_write` access for a texture buffer with the pixel format. [1]

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

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 [2]
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 feature by inspecting its `MTLDevice.readWriteTextureSupport` property at runtime.
2. GPUs that support texture atomics (see feature availability by GPU family) also support atomics in read/write texture buffers with this pixel format.

Tensor limits by data type

Tensor limits	Rank [1]	Row size alignment [2]	Row stride alignment [3]	Slice offset and size alignment [4]	Buffer offset alignment [5]	Supports block scaling
2-bit tensor format data types						
`Int2`	1 … 16	32 elements	512 elements	4 elements	128 bytes	Yes
`UInt2`	1 … 16	32 elements	512 elements	4 elements	128 bytes	Yes
4-bit tensor format data types						
`MetalFloat4E2M1`	1 … 16	32 elements	256 elements	2 elements	128 bytes	Yes
`Int4`	1 … 16	32 elements	256 elements	2 elements	128 bytes	Yes
`UInt4`	1 … 16	32 elements	256 elements	2 elements	128 bytes	Yes
8-bit tensor format data types						
`MetalFloat8E5M2`	1 … 16	32 elements	128 elements	1 element	128 bytes	Yes
`MetalFloat8E4M3`	1 … 16	32 elements	128 elements	1 element	128 bytes	Yes
8-bit tensor ordinary data types						
`Int8`	0 … 16	1 element	1 element	1 element	1 byte	Yes
`UInt8`	0 … 16	1 element	1 element	1 element	1 byte	Yes
16-bit tensor ordinary data types						
`Float16`	0 … 16	1 element	1 element	1 element	2 bytes	No
`BFloat16`	0 … 16	1 element	1 element	1 element	2 bytes	No
`Int16`	0 … 16	1 element	1 element	1 element	2 bytes	No
`UInt16`	0 … 16	1 element	1 element	1 element	2 bytes	No
32-bit tensor ordinary data types						
`Float32`	0 … 16	1 element	1 element	1 element	4 bytes	No
`Int32`	0 … 16	1 element	1 element	1 element	4 bytes	No
`UInt32`	0 … 16	1 element	1 element	1 element	4 bytes	No
Tensor block scale data types [6]						
`MetalFloat8UE8M0`	N/A	N/A	N/A	1 element	128 bytes	N/A

1. Create `MTLTensorExtents` with `rank` in this range. The range is inclusive of both ends.
2. When tensor rank is greater than 0, set `dimensions.extents[0]` to a multiple of this value. Always set `strides.extents[0]` to 1.
3. When tensor rank is greater than 1, set `strides.extents[1]` to a multiple of this value. If the tensor usage includes `MTLTensorUsage.machineLearning`, set it to a value that is also a multiple of 64 bytes divided by the element size.
4. Set `extents[0]` of origin and dimension parameters of `copy`, `MTLTensor.getBytes` and `MTLTensor.replace` to a multiple of this value.
5. When creating a tensor or tensor plane over an existing buffer, set the buffer offset to a multiple of this value. If the tensor usage includes `MTLTensorUsage.machineLearning`, set it to a value that is also a multiple of 64 bytes.
6. Use these formats for auxiliary tensor planes with type `MTLTensorPlaneType.scales`. The rank, row size and row stride of the scales plane are implied by the configuration of the data plane.

Apple Inc.

Apple Inc.
One Apple Park Way
Cupertino, CA 95014