Metal Feature Set Tables

Developer

Metal GPUs (Apple silicon)

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

Metal GPUs (Intel Mac)

GPU	Metal version	Mac family [1]
AMD 500-series	Metal	Mac2
AMD Vega	Metal 3	Mac2
AMD 5000-series	Metal 3	Mac2
AMD 6000-series	Metal 3	Mac2
Intel UHD Graphics 630	Metal 3	Mac2
Intel Iris Plus Graphics	Metal 3	Mac2

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

For Mac devices with Apple silicon, the `MTLDevice` instance for the Apple GPU reports that it also supports `Mac2` GPU family because the devices support the union of both feature families.

Metal feature availability by GPU family

GPU family [1]	Metal	Apple	Mac
Feature	**Available in family**		
MetalKit	Metal3	Apple2	Mac2
Metal performance shaders	Metal3	Apple2	Mac2
Programmable blending	Metal4	Apple2	—
Separate depth and stencil attachments	Metal4	Apple2	—
PVRTC pixel formats	—	Apple2	—
EAC/ETC pixel formats	Metal4	Apple2	—
ASTC pixel formats	Metal4	Apple2	—
BC pixel formats [2]	—	Apple9	Mac2
Compressed volume texture formats	Metal3	Apple3	Mac2
Extended range pixel formats	Metal4	Apple3	—
Wide color pixel format	Metal3	Apple2	Mac2
Depth-16 pixel format	Metal3	Apple2	Mac2
Linear textures	Metal3	Apple2	Mac2
MSAA depth resolve	Metal3	Apple3	Mac2
Array of textures (read)	Metal3	Apple3	Mac2
Array of textures (write)	Metal3	Apple6	Mac2
Cube map texture arrays	Metal3	Apple4	Mac2
Stencil texture views	Metal3	Apple2	Mac2
Array of samplers	Metal3	Apple3	Mac2
Sampler maximum anisotropy	Metal3	Apple2	Mac2
Sampler LOD clamp	Metal3	Apple2	Mac2
MTLSamplerState support for comparison functions	Metal3	Apple3	Mac2
16-bit unsigned integer coordinates	Metal3	Apple2	Mac2
Border color	Metal3	Apple7	Mac2
Counting occlusion query	Metal3	Apple3	Mac2
Base vertex/instance drawing	Metal3	Apple3	Mac2
Layered rendering	Metal3	Apple5	Mac2
Layered rendering to multisample textures	Metal3	Apple7	Mac2
Memoryless render targets	Metal4	Apple2	—
Dual-source blending	Metal3	Apple2	Mac2
Combined MSAA store and resolve action	Metal3	Apple3	Mac2
MSAA blits	Metal3	Apple2	Mac2
Programmable sample positions	Metal3	Apple2	Mac2
Deferred store action	Metal3	Apple2	Mac2
Texture barriers	—	—	Mac2
Memory barriers [3]	Metal3	Apple3	Mac2
Indirect command buffer support for memory barriers (compute)	Metal3	Apple3	Mac2
Indirect command buffer support for memory barriers (rendering)	Metal4	Apple9	—
Tessellation	Metal3	Apple3	Mac2
Indirect tessellation arguments	Metal3	Apple5	Mac2
Tessellation in indirect command buffers	Metal3	Apple5	Mac2
Resource heaps	Metal3	Apple2	Mac2
Function specialization	Metal3	Apple2	Mac2
Read/write buffers in functions	Metal3	Apple3	Mac2
Read/write textures in functions	Metal3	Apple4	Mac2
Extract, insert, and reverse bits	Metal3	Apple2	Mac2
SIMD barrier	Metal3	Apple2	Mac2
Indirect draw and dispatch arguments	Metal3	Apple3	Mac2
Argument buffers tier 1	Metal3	Apple2	Mac2
Argument buffers tier 2	Metal3	Apple6	Mac2
Indirect command buffers (rendering)	Metal3	Apple3	Mac2
Indirect command buffers (compute)	Metal3	Apple3	Mac2
Uniform type	Metal3	Apple2	Mac2
Imageblocks	Metal4	Apple4	—
Tile shaders	Metal4	Apple4	—
Imageblock sample coverage control	Metal4	Apple4	—
Post-depth coverage	Metal4	Apple4	—
Quad-scoped permute operations	Metal3	Apple4	Mac2

GPU family [1]	Metal	Apple	Mac
Quad-scoped ballot operation	Metal3	Apple6	Mac2
Quad-scoped reduction operations	Metal3	Apple7	Mac2
SIMD-scoped permute operations	Metal3	Apple6	Mac2
SIMD-scoped reduction operations	Metal3	Apple7	Mac2
SIMD-scoped matrix multiply operations	Metal4	Apple7	—
Raster order groups [4]	Metal3	Apple4	Varies
Nonuniform threadgroup size	Metal3	Apple4	Mac2
Multiple viewports	Metal3	Apple5	Mac2
Device notifications	—	—	Mac2
Stencil feedback	Metal3	Apple5	Mac2
Stencil resolve	Metal3	Apple5	Mac2
Nonsquare tile dispatch	Metal4	Apple5	—
Texture swizzle	Metal3	Apple2	Mac2
Placement heap	Metal3	Apple2	Mac2
Primitive ID	Metal3	Apple7	Mac2
Barycentric coordinates [5]	Metal4	Apple7	Varies
Read/write cube map textures in functions	Metal3	Apple4	Mac2
Sparse textures [6]	Metal4	Apple6	—
Sparse depth and stencil textures [7]	—	Apple8	—
Variable rasterization rate [8]	Metal4	Apple6	Varies
Vertex amplification [9]	Metal4	Apple6	Varies
64-bit integer math	Metal3	Apple3	—
Lossy texture compression	—	Apple8	—
SIMD shift and fill	—	Apple8	—
Render dynamic libraries	Metal4	Apple6	—
Compute dynamic libraries	Metal3	Apple6	Mac2
Mesh shading	Metal3	Apple7	Mac2
Indirect mesh draw arguments	—	Apple9	—
Indirect command buffers containing mesh draws	—	Apple9	—
MetalFX spatial upscaling	Metal3	Apple3	Mac2
MetalFX temporal upscaling	Varies	Apple7	—
MetalFX frame interpolation	Metal4	Apple5	—
MetalFX denoised upscaling	—	Apple9	—
Fast resource loading	Metal3	Apple2	Mac2
Ray tracing in compute pipelines [10]	Metal3	Apple6	Varies
Ray tracing in render pipelines [11]	Metal4	Apple6	—
Floating-point atomics	Metal3	Apple7	Mac2
Texture atomics	Metal3	Apple6	Mac2
64-bit atomics [12]	—	Apple9	—
Query texture LOD [13]	—	Apple8	—
Binary archives	Metal3	Apple3	Mac2
Function pointers in compute pipelines [14]	Metal3	Apple6	Varies
Function pointers in render pipelines [11]	Metal4	Apple6	—
Depth sample compare bias and gradient	Metal4	Apple2	—
Nonprivate depth stencil textures	Metal4	Apple2	—
Dynamic stride for attribute buffers	Metal3	Apple4	Mac2
`MTLAttributeFormat.floatRGB9E5` and `.floatRG11B10`	Metal3	Apple5	Mac2
`MTLDataType.bfloat` (brain float) scalar and vector cases	Metal3	Apple6	Mac2
Relaxed math	Metal4	Apple4	—
Global built-ins and bindings	Metal4	Apple6	—
Memory coherence for textures and buffers in shaders	Metal4	Apple6	—
Per-pipeline shader validation	Metal4	Apple6	—
Shader logging	Metal4	Apple6	—
Residency sets	Metal4	Apple6	—
Acceleration structures containing row-major matrices	—	Apple9	—
Ray tracing with per-component motion interpolation	—	Apple9	—
Direct access to on-chip ray-intersection result storage	—	Apple9	—
Fragment visibility count accumulation [15]	Metal4	Apple7	—
Argument tables	Metal4	Apple7	—
Command allocators	Metal4	Apple7	—

GPU family [1]	Metal	Apple	Mac
Decoupled command queues and command buffers	Metal4	Apple7	—
Texture view pools	Metal4	Apple7	—
Command barriers	Metal4	Apple7	—
Placement sparse buffers [16]	—	Apple8	—
Placement sparse textures [16]	—	Apple8	—
Dedicated compilation contexts	Metal4	Apple7	—
Pipeline dataset serialization	Metal4	Apple7	—
Flexible render pipeline state	Metal4	Apple7	—
Color attachment mapping [15]	Metal4	Apple7	—
Machine learning encoding	Metal4	Apple7	—
Tensors	Metal4	Apple7	—
Performance counter heaps	Metal4	Apple7	—
Address-driven acceleration structure builds	—	Apple9	—
Acceleration structure build options	—	Apple9	—
Intersection function buffers	—	Apple9	—
Sampler minimum and maximum reduction	—	Apple10	—
Sampler LOD bias	—	Apple10	—
Access to pre-raster per-vertex values	—	Apple10	—
Depth bounds testing	—	Apple10	—
Indirect command buffer support for raster and depth/stencil state	—	Apple10	—
Atomics on cube map and cube map array textures	—	Apple10	—
Universal texture compression	—	Apple10	—

1. 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. 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. 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.
7. 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.
8. 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.
9. 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.
10. Some GPU devices in the `Mac2` family support ray tracing in compute pipelines. You can check an individual GPU's support for this feature 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 Metal IR linking through `MTLLinkedFunctions.privateFunctions` in render pipelines using mesh shading.
12. Some GPU devices in the `Apple8` family support 64-bit atomic minimum and maximum using `ulong`, on both buffers and textures. You can check an individual GPU's support for this feature by verifying it supports both the `Mac2` and `Apple8` families by separately passing each to the `MTLDevice.supportsFamily(_:)` method. As of `Apple9` all GPUs have support.
13. 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.
14. Some GPU devices in the `Mac2` family support function pointers in compute pipelines. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.supportsFunctionPointers` property at runtime.
15. GPU devices supporting fragment visibility count accumulation and color attachment mapping features support those features in both Metal3 and Metal4 command encoding models.
16. 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]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
Function arguments	**Function arguments**											
Maximum number of vertex attributes, per vertex descriptor [2]	31	31	31	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	31	31	31
Maximum number of entries in the texture argument table, per graphics or kernel function [2]	128	128	31	31	96	96	128	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	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	31	31	31
Maximum number of constant buffer arguments in vertex, fragment, tile, or kernel functions [2]	14	31	31	31	31	31	31	31	31	31	31	14
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	4 KB	4 KB	4 KB
Maximum threads per threadgroup [5]	1024	1024	512	512	1024	1024	1024	1024	1024	1024	1024	1024
Maximum total threadgroup memory allocation	32 KB	32 KB	16,352 B	16 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB
Maximum explicit image block allocation [6]	Not available	32 KB	Not available	Not available	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB	Not available
Maximum implicit image block allocation [6]	Not available	128 KB	Not available	Not available	128 KB	128 KB	128 KB	128 KB	128 KB	256 KB	256 KB	Not available
Threadgroup memory length alignment	16 B	16 B	16 B	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]	32	124	60	60	124	124	124	124	124	124	124	32
Maximum number of input components to a fragment function. (Declare with the `[[stage_in]]` qualifier.) [7]	124	124	60	60	124	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	65,536	65,536	65,536
Maximum tessellation factor	64	64	Not available	16	16	64	64	64	64	64	64	64
Maximum number of viewports and scissor rectangles, per vertex function	16	16	1	1	1	16	16	16	16	16	16	16
Maximum number of raster order groups, per fragment function	8	8	Not available	Not available	8	8	8	8	8	8	8	8
Minimum alignment of buffer layout descriptor stride and attribute descriptor offset	4 B	1 B	4 B	4 B	4 B	1 B	1 B	1 B	1 B	1 B	1 B	4 B
Maximum size of buffer layout descriptor stride	4 KB	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit	4 KB
Argument buffers [8]	**Argument buffers**											
Maximum number of buffers you can access, per stage, from an argument buffer	No limit	No limit	31	31	96	96	No limit	No limit	No limit	No limit	No limit	No limit
Maximum number of textures you can access, per stage, from an argument buffer	1 M	1 M	31	31	96	96	1 M	1 M	1 M	1 M	1 M	1 M
Maximum number of samplers you can access, per stage, from an argument buffer	1024	1024	16	16	16	16	128	996	996	500 K	500 K	1024

GPU family [1]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
Resources	**Resources**											
Minimum constant buffer offset alignment	32 B	4 B	4 B	4 B	4 B	4 B	4 B	4 B	4 B	4 B	4 B	32 B
Maximum 1D texture width	16,384 px	16,384 px	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	16,384 px
Maximum 2D texture width and height	16,384 px	16,384 px	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	16,384 px
Maximum cube map texture width and height	16,384 px	16,384 px	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	16,384 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	2048 px	2048 px	2048 px
Maximum texture buffer width [9]	256 M px	256 M px	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	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	2048	2048	2048
Buffer alignment for copying an existing texture to a buffer	256 B	16 B	64 B	16 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	32 KB	32 KB	32 KB	No limit
Maximum number of sample buffers	32	32	32	32	32	32	32	32	32	32	32	No limit
Maximum number of residency sets per queue	32	32	Not available	Not available	Not available	Not available	32	32	32	32	32	32
Maximum number of residency sets per buffer	32	32	Not available	Not available	Not available	Not available	32	32	32	32	32	32
Maximum indirect command buffer length	16,384 commands	No limit	Not available	No limit	No limit	No limit	No limit	No limit	No limit	No limit	No limit	16,384 commands
Render targets	**Render targets**											
Maximum number of color render targets per render pass descriptor	8	8	8	8	8	8	8	8	8	8	8	8
Maximum size of a point primitive	511	511	511	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	64 B	Not available	Not available	64 B	64 B	64 B	64 B	64 B	64 B	64 B	Not available
Maximum implicit image block size, per pixel, per sample, when using multiple color render targets	Not available	128 B	32 B	32 B	64 B	64 B	64 B	128 B	128 B	128 B	128 B	Not available
Maximum visibility query offset	256 KB	256 KB	65,528 B	65,528 B	65,528 B	65,528 B	65,528 B	256 KB	256 KB	256 KB	256 KB	256 KB
Maximum sample count in render passes with MSAA	4	4	4	4	4	4	4	4	4	4	8	4
Maximum tile size in render passes without MSAA	Not available	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	32 x 32	Not available
Maximum tile size in render passes with 2x MSAA	Not available	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	32 x 32	Not available
Maximum tile size in render passes with 4x MSAA	Not available	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	32 x 16	Not available
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	Not available	Not available	16 x 16	Not available
Feature limits	**Feature limits**											

GPU family [1]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
Maximum number of fences	32,768	32,768	32,768	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	8192	8192	8192
Maximum vertex count for vertex amplification [10]	Varies	8	Not available	Not available	Not available	Not available	2	8	8	8	8	Varies
Maximum threadgroups per object shader grid	1024	No limit	Not available	Not available	Not available	Not available	Not available	No limit	No limit	No limit	No limit	1024
Maximum threadgroups per mesh shader grid [11]	1024	1024	Not available	Not available	Not available	Not available	Not available	1024	1024	1,048,575	4,194,303	1024
Maximum payload in mesh shader pipeline [12]	16,384 B	16,384 B	Not available	Not available	Not available	Not available	Not available	16,384 B	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 [13]	32	32	Not available	Not available	Not available	Not available	32	32	32	32	32	32
Largest number of levels a ray-tracing intersection query can traverse in an acceleration structure [13]	16	16	Not available	Not available	Not available	Not available	16	16	16	16	16	16
Maximum texture view pool entries	Not available	128 million	Not available	Not available	Not available	Not available	Not available	128 million	128 million	256 million	256 million	Not available
Maximum supported tensor rank	Not available	16	Not available	Not available	Not available	Not available	Not available	16	16	16	16	Not available
Maximum supported tensor stride at dimension index 0 for machine learning encoder usage	Not available	1 element	Not available	Not available	Not available	Not available	Not available	1 element	1 element	1 element	1 element	Not available
Minimum alignment of tensor stride at dimension index 1 for machine learning encoder usage	Not available	64 B	Not available	Not available	Not available	Not available	Not available	64 B	64 B	64 B	64 B	Not available
Maximum performance counter heaps (per process)	Not available	32	Not available	Not available	Not available	Not available	Not available	32	32	32	32	Not available
Minimum alignment of intersection function buffer	Not available	64 B	Not available	Not available	Not available	Not available	Not available	64 B	64 B	64 B	64 B	Not available
Minimum alignment of intersection function buffer stride	Not available	8 B	Not available	Not available	Not available	Not available	Not available	8 B	8 B	8 B	8 B	Not available
Maximum size of intersection function buffer stride	Not available	4096 B	Not available	Not available	Not available	Not available	Not available	4096 B	4096 B	4096 B	4096 B	Not available

1. See `MTLGPUFamily` for each GPU family's enumeration constant.
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. 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.
11. Mesh shaders can use up to 4 GB of payload and mesh geometry per draw for devices in the `Apple7` and `Apple8` GPU families.
12. 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.
13. 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 `Mac2`, `Metal3`, and `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]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
Ordinary 8-bit pixel formats	Texture capabilities for **ordinary 8-bit pixel formats** by GPU family											
`A8Unorm` [2,3]	All	All	Filter	All	All	All	All	All	All	All	All	All
`R8Unorm` [2]	All	All	All	All	All	All	All	All	All	All	All	All
`R8Unorm_sRGB`	Not available	All	All	All	All	All	All	All	All	All	All	Not available
`R8Snorm`	All	All	All	All	All	All	All	All	All	All	All	All
`R8Uint` [2] `R8Sint` [2]	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
Ordinary 16-bit pixel formats	Texture capabilities for **ordinary 16-bit pixel formats** by GPU family											
`R16Unorm` `R16Snorm`	All	All	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	All	All	All	All	All	All	All	All
`R16Uint` [2] `R16Sint` [2]	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
`R16Float` [2]	All	All	All	All	All	All	All	All	All	All	All	All
`RG8Unorm`	All	All	All	All	All	All	All	All	All	All	All	All
`RG8Unorm_sRGB`	Not available	All	All	All	All	All	All	All	All	All	All	Not available
`RG8Snorm`	All	All	All	All	All	All	All	All	All	All	All	All

GPU family [1]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
`RG8Uint` `RG8Sint`	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
Packed 16-bit pixel formats [4]	Texture capabilities for **packed 16-bit pixel formats** by GPU family											
`B5G6R5Unorm` `A1BGR5Unorm` [5] `ABGR4Unorm` [5] `BGR5A1Unorm`	Not available	Filter Color MSAA Resolve Blend Sparse	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	Not available
Ordinary 32-bit pixel formats	Texture capabilities for **ordinary 32-bit pixel formats** by GPU family											
`R32Uint` [2] `R32Sint` [2]	Atomic Write Color MSAA	Atomic Write Color MSAA Sparse	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	Atomic Write Color MSAA
`R32Float` [2,6]	Write Color MSAA Blend	Write Color MSAA Blend Sparse	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	All
`RG16Unorm` `RG16Snorm`	All	All	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	All	All	All	All	All	All	All	All
`RG16Uint` `RG16Sint`	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
`RG16Float`	All	All	All	All	All	All	All	All	All	All	All	All
`RGBA8Unorm` [2]	All	All	All	All	All	All	All	All	All	All	All	All
`RGBA8Unorm_sRGB`	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All	All	All	Filter Color MSAA Resolve Blend
`RGBA8Snorm`	All	All	All	All	All	All	All	All	All	All	All	All

GPU family [1]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
`RGBA8Uint` [2] `RGBA8Sint` [2]	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
`BGRA8Unorm`	All	All	All	All	All	All	All	All	All	All	All	All
`BGRA8Unorm_sRGB`	Filter Color MSAA Resolve Blend	All	All	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`	All	All	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All	All
`BGR10A2Unorm`	All	All	All	All	All	All	All	All	All	All	All	All
`RGB10A2Uint`	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
`RG11B10Float` [4]	All	All	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All	All
`RGB9E5Float` [4]	Filter	All	Filter Color MSAA Resolve Blend	All	All	All	All	All	All	All	All	Filter
Ordinary 64-bit pixel formats	Texture capabilities for **ordinary 64-bit pixel formats** by GPU family											
`RG32Uint` [7] `RG32Sint`	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
`RG32Float` [6]	Write Color MSAA Blend	Write Color MSAA Blend Sparse	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	All

GPU family [1]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
`RGBA16Unorm` `RGBA16Snorm`	All	All	Filter Write Color MSAA Blend	Filter Write Color MSAA Blend	All	All	All	All	All	All	All	All
`RGBA16Uint` [2] `RGBA16Sint` [2]	Write Color MSAA	Write Color MSAA Sparse	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	Write Color MSAA
`RGBA16Float` [2]	All	All	All	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 MSAA	Write Color MSAA Sparse	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	Write Color MSAA
`RGBA32Float` [2,6]	Write Color MSAA Blend	Write Color MSAA Blend Sparse	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	All
Compressed pixel formats [4]	Texture capabilities for **compressed pixel formats** by GPU family											
PVRTC pixel formats [8]	Not available	Filter Sparse	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
EAC/ETC pixel formats	Not available	Filter Sparse	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
ASTC pixel formats	Not available	Filter Sparse	Filter	Filter	Filter	Filter	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
HDR ASTC pixel formats	Not available	Filter Sparse	Not available	Not available	Not available	Not available	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Filter Sparse	Not available
BC pixel formats	Varies [9]	Varies [9]	Not available	Not available	Not available	Not available	Not available	Varies [9]	Varies [9]	Filter Sparse	Filter Sparse	Filter
YUV pixel formats [4]	Texture capabilities for **YUV pixel formats** by GPU family											
`GBGR422` `BGRG422`	Filter	Filter	Filter	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 Resolve	Filter MSAA Resolve Sparse [10]	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	Filter MSAA Resolve
`Depth32Float` [6]	MSAA Resolve	MSAA Resolve Sparse [10]	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve Sparse [10]	MSAA Resolve Sparse	Filter MSAA Resolve Sparse	Filter MSAA Resolve Sparse	Filter MSAA Resolve

GPU family [1]	Metal3	Metal4	Apple2	Apple3	Apple4	Apple5	Apple6	Apple7	Apple8	Apple9	Apple10	Mac2
`Stencil8`	Not available	MSAA Resolve Sparse [10]	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve Sparse [10]	MSAA Resolve Sparse	MSAA Resolve Sparse	MSAA Resolve Sparse	Not available
`Depth24Unorm_Stencil8` [11]	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Filter MSAA Resolve
`Depth32Float_Stencil8`	MSAA Resolve	MSAA Resolve	MSAA	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	MSAA Resolve	Filter MSAA Resolve	Filter MSAA Resolve	Filter MSAA Resolve
`X24_Stencil8`	Not available	Not available	Not available	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	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	Not available	All	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. 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.
11. Some GPUs support `MTLPixelFormat.depth24Unorm_stencil8`. You can check an individual GPU's support for this feature by inspecting its `MTLDevice.isDepth24Stencil8PixelFormatSupported` 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. [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 feature 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.

Apple Inc.

Apple Inc.
One Apple Park Way
Cupertino, CA 95014