# Graphics and Music Challenges for Classic Computer Applications

Peter Occil

This version of the document is dated 2024-06-29.

The following are three challenges I make to the computer community, relating to:

- Graphics for retro-style game development.
- MIDI music synthesis for retro-style games and apps.
- Tileable wallpapers with limited colors and resolution.

All three serve to arouse nostalgia among 90s computer users.

## 1 Graphics Challenge for Retro Style Games

An interesting challenge for game developers.

Limit 3D graphics to the following:

1. No more than 2048 polygons can be displayed at a time.
    - A polygon is either a triangle, a convex quadrilateral, or a line segment.
    - Each vertex of the polygon must point to a vertex from the vertex list described below.
    - Each polygon can be translucent and/or wireframed.
2. Polygons undergo perspective-correct rendering.
3. The maximum number of vertices can be used at a time is 3 times the maximum number of polygons.
    - A vertex consists of an XYZ position, an XY texture coordinate, and an RGB vertex color.
    - For each color, the red component is 5 bits, the green, 5 bits, and the blue, 5 bits.
4. Textures must have the same color format as vertex colors, or may employ a 4-, 16-, or 256-color palette with that color format. Texture rendering supports flips and repeats on either or both axes. A texture's maximum image size is 192 kibibytes.
5. Depth tests, clear colors, and fog colors are supported.
6. The 3D graphics buffer's resolution is the same as the screen resolution.

Limit 2D graphics to the following:

1. Up to three tile-based 2D layers can be displayed at a time. If 3D graphics are not being displayed, a fourth tile-based 2D layer can also be displayed. Otherwise, a layer for the 3D graphics can be displayed.
2. There are sixteen palettes of 16 colors each (using the color format for vertex colors).
3. Each tile is 8×8 pixels and uses the colors of one of the sixteen palettes just described.
4. The 2D and 3D layers may contain transparent parts.
5. One of the 2D layers can undergo a 2D affine transformation.
6. Separate from layers, 2D sprites can be displayed. No more than 128 sprites may be displayed at a time. Each sprite may be tile-based or bitmap-based and must have a width and height of no more than 64 pixels each. Sprites may contain transparent parts.

General:

- The 3D graphics layer, if any, can be alpha blended with the 2D graphics layers in any order.

- 256×192 screen resolution (256 pixels wide by 192 pixels high) with up to 60 frames per second, or 256×384 screen resolution with up to 30 frames per second.
- A game may limit the amount of graphics memory (akin to VRAM) to a certain maximum size, say, 2048 kibibytes. This does not limit the size or number of graphics assets a game can have.
- Music: Standard MIDI files (SMF) only. The files should be rendered using a cross-platform open-source software synthesizer (see next section), using either FM or wavetable synthesis.[1] As much as possible, instruments should match their meanings in the General MIDI System level 1.

A game might use a different resolution than shown. In that case, the maximum allowed number of polygons and vertices and the maximum texture size, sprite size, and sprite count, as well as the maximum graphics memory size, if any, will change in proportion to the new resolution. (For example, if the resolution is 640×480 with up to 60 frames per second, these maximums are multiplied by 6.25 = (640×480) / (256×192). Other resolutions used in classic games include 320×200, 320×240, 640×350, and 512×384.)

These limitations were inspired by the graphics limitations of classic handheld game consoles.

A game may impose further resource limits to the specifications given here (for example, to reduce the maximum number of 3D polygons, to disallow polygons, or to reduce the number of colors per tile allowed). I would be interested in knowing about these limitations that a new game that adopts this document decides to impose. I would also be interested in learning about a free and open-source graphics library that implements this specification.[2]

# 2 Building a Public-Domain music synthesis library and instrument banks

To improve support for MIDI (Musical Instrument Digital Interface) music playback in open-source and other applications, I challenge the community to write the following items, all of which must be released to the public domain (Creative Commons Zero).

- A cross-platform open-source library for *software* synthesis of MIDI data stored in standard MIDI files (SMF, .mid), using instrument sound banks in SoundFont 2 (.sf2), downloadable sounds (.dls), and in OPL2, OPL3, and other FM synthesis sound banks, and possibly also in Timidity++/UltraSound patch format (.cfg, .pat). (Similar to *Fluidsynth*, but in the public domain. Instrument sound banks are files that describe how to render MIDI instruments as sound.) In addition, the source code in the non-public-domain *foo_midi*, *libADLMIDI*, *libOPNMIDI*, and *OPL3BankEditor* may be useful here, but review their licenses first.)
  - The library should support popular loop-point conventions found in MIDI files.
  - The library should support seeking of MIDI files such that a pause and resume function can be offered by a media player.
- An instrument sound bank for wavetable synthesis of all instruments and drum noises in the General MIDI System level 1 specification.
  - Instruments should correspond as closely as possible to those in that specification, but should be small in file size or be algorithmically generated.
  - Instruments can be generated using the public-domain single-cycle wave forms found in the AdventureKid Wave Form collection, found at: **AKWF-FREE**[3].

[1] I note that it's possible to write an FM software synthesizer supporting every MIDI instrument in less than 1024 kibibytes of code.

[2] Especially if the library is self-contained and implements the specification with as little source code as possible. It would not be within the spirit of this document to, say, display more polygons or vertices at a time than the maximum allowed using programming tricks, but any such tricks should not be hardware-accelerated. An example of a 2D library that follows the spirit of this specification, even though it doesn't necessarily meet its requirements exactly, is called ***Tilengine***. https://github.com/megamarc/Tilengine

[3] https://github.com/KristofferKarlAxelEkstrand/AKWF-FREE

  - The samples for each instrument are preferably generated by an algorithm, such as one that renders the instrument's tone in the frequency domain. An example of this is found in `com.sun.media.sound.EmergencySoundbank`[4], which however is licensed under the GNU General Public License version 2 rather than public domain (Creative Commons Zero).
  - The instrument sound bank should be in either SoundFont 2 (.sf2) or downloadable sounds (.dls) format.
    * A sound bank of decent quality in either format is about 4 million bytes in size.
  - The volume of all instruments in the sound bank should be normalized; some instruments should not sound louder than others.
- An instrument sound bank for FM synthesis of all instruments and drum noises in the General MIDI System level 1 specification. Instruments should correspond as closely as possible to those in that specification.

# 3 Classic Wallpaper Challenge

See the "**peteroupc/classic-wallpaper**[5]" repository for a challenge on creating tileable desktop wallpapers with a limited palette of colors and a limited pixel size — such wallpapers are getting ever harder to find because desktop backgrounds today tend to cover the full computer screen, to employ thousands of colors, and to have a high-definition resolution (1920×1080 or larger).

# 4 License



---

[4] https://github.com/apple/openjdk/blob/xcodejdk14-release/src/java.desktop/share/classes/com/sun/media/sound/EmergencySoundbank.jav
[5] https://github.com/peteroupc/classic-wallpaper
[6] https://creativecommons.org/publicdomain/zero/1.0/