**Engine Subsystems**

Entries highlighted in green have been completed. Entries in yellow are incomplete; non-highlighted entries have not been worked on. Optional items are in *italics*.

1. **Platform Independence**
   1. Datatypes
   2. OS Dependent Wrapper
      1. File System. Use existing library if possible.
         1. File Open/Close
         2. Directory Enumeration
      2. Timer
      3. Input Device I/O (Raw Input). No existing library found.
         1. Input API Specification
      4. Supported OSes
         1. Windows
         2. *Linux*
         3. *OS X*
   3. Threading System. Use existing library if possible. May be OS dependent, may not. Very hard to tell right now.
   4. Graphics Wrapper
      1. Function Loader
      2. Function Wrapper
      3. Supported Frameworks. Use existing libraries if possible.
         1. OpenGL
         2. *DirectX*
   5. Audio Wrapper. Check <https://github.com/RustAudio> .
      1. Playback / Manager
      2. 3D Audio
      3. *DSP & Effects (distortion, low priority)*
      4. Supported Platforms:
         1. Windows (choose a library)
         2. *Linux*
         3. *OS X*
   6. Physics System/Wrapper. May need to hook to C++ library if none in Rust is viable.
2. **Core**
   1. Memory Allocation
      1. Allocator Specification
      2. Implementations
         1. Stock allocator. May be slow, but requires no new work. we’ll see as we go on.
         2. *Special pool/freelist allocator?* May be too fancy at the moment.
   2. Module Startup & Shutdown
   3. Assertion System. Use existing library if possible.
   4. Debug Logging. Use existing library if possible.
   5. RNG. Use existing library if possible.
   6. Math Libraries. Use existing library if possible.
   7. Strings & Hashed Strings. Use existing library if possible.
   8. Engine Configuration
      1. Server Variables
      2. Client Variables
   9. Profiler/Stat Gatherer. Use existing library if possible.
   10. Object Handles & IDs. Intend a DoD design; this implies a component system and sparse arrays.
       1. Component System
       2. Sparse Array Implementation
   11. *Curves & Surfaces*
   12. *Reflection & Serialization*
   13. *Localization*
   14. *Async File I/O*
   15. *Movie Player/Replay System*
   16. *Debug Menu/Console*
3. **Resource Manager**
   1. Resource Archive Format. Stores all related assets under one file.
      1. Resource Archive Specification
         1. Compression Method. Use existing library if possible.
         2. Asset Specification
      2. Asset Management
         1. Add
         2. Remove
         3. Move
      3. Asset Loading
      4. Asset Packing
   2. Custom Formats
      1. Models. Models need a single binary format that we can quickly load in.
         1. Spec for Model Format. Models must support at the bare minimum:
            1. Vertices
            2. Normals
            3. Diffuse Textures
            4. Materials
            5. *Vertex Animation?*
   3. Resource Exporters
      1. Models
         1. Blender plugin. Uses Python.
         2. *Maya plugin*. Uses MEL.
   4. Resource Importers
      1. Audio. Use existing libraries if possible.
         1. WAV
         2. OGG
         3. *MP3*
      2. *Textures.* Usually we save as PNG or JPG, but we need to convert that to a 24-bit or compressed texture for faster loading.
      3. *Fonts*. Has a system in the form of freetype; still subject to change, however.
      4. *Colliders*
      5. *Physics Params*
      6. *Maps*
4. **Input**
   1. Control Abstraction
      1. Keyboard
      2. Button
      3. Axis
   2. I/O Filtering
      1. Deadzones
5. **Collision Detection/Physics**

Try to use an existing library for the backend.

* 1. Forces & Constraints
  2. Raycasting
  3. Rigid bodies
  4. Collision bodies
  5. Phantoms
  6. Physics/Collision world
  7. *Shapecasting*

1. **Low-Level Renderer**
   1. Materials
   2. Shaders
      1. Diffuse Shading
         1. Multiple Light Support
         2. Choose a Lighting Model
      2. *Special Effects*
         1. *Glow?*
         2. *Blur?*
   3. Camera. Need a way to represent the viewer’s viewpoint.
   4. Primitive Submission System. Model draw requests must be broken down into individual draw calls eventually, and this is what does that.
   5. Viewport/Virtual Screen
   6. Texture & Surface Management. Handle loading and unloading textures to the GPU.
   7. Text & Fonts. Render text straight to the screen.
   8. Debug Primitives. Draw a primitive at the requested transform.

(From LeEK 1: Cylinders and toruses don't have a drawing function, and since 6.4 is not implemented the debug calls force the system to switch to the debug shader and setup matrices every time a debug primitive needs to be drawn. Not a major concern, however, since 6.4 needs to be implemented along with Section 9 anyway.)

* 1. Device Interface

1. **Scene Graph**
   1. Space Subdivision. Octree as in LeEK 1 for now.
   2. *LOD*
   3. *Occlusion*
2. **Front End**
   1. Game GUI & Menu Screens. HUGE CAN OF WORMS! May want to use existing library.
      1. GUI Layout API
      2. Images
      3. Text
      4. Buttons
         1. Button Labels
      5. Radio Buttons
      6. Drop-down Boxes
      7. Sliders
      8. Text Boxes
         1. Multiline
         2. Scrolling
   2. *Attract Mode*
3. **Gameplay Base**
   1. Game Flow System
   2. Scripting
      1. Incomplete, see LeEK 1: “Scripting Notes”.
   3. Static World
   4. Dynamic Object Model (Actor and Components)
   5. Agent System (AI hooks)
   6. Event / Messaging System
   7. World Loading