Vykreslovací engine
Vykreslovací engine vytváří 2D obrázek ze vstupních dat o scéně. 2D obrázek si můžeme představit jako 2 rozměrné pole, kde prvku se říká pixel. Ten drží nějakou hodnotu.

Vykreslovací engine je většinou součástí nějakého většího systému, například herního enginu. *Herní engine* je vícero enginů, například ještě fyzický, pro simulování kolizí objektů, apod. pro interaktivních zážitků.

Real-time
Vykreslovací engine, který pracuje v reálném čase znamená, že tyto obrázky dokáže vykreslovat takovou rychlostí, že pro lidské oko se jeví jako pohyb.

Scéna
Vykreslovací engine potřebuje vstup, který bude říkat, co má vykreslit. Takovým vstupem bude scéna, která bude obsahovat množinu objektů a světel.

Objekt zadefinujeme jako

Projekce
Nejprve musíme množinu polygonů zobrazit na dvojrozměrný prostor. To uděláme tzv. projekční maticí.

Grafická API

Vulkan
Vulkan je moderní, nízko-úrovňová grafická API. Oproti předchůdcům, jako DirectX 11 nebo OpenGL, dává daleko větší kontrolu nad celým procesem renderování. Mnoho funkcí a optimalizací do té doby dělali samotné ovladače. Vulkan je pouze API, a tím pádem to, jak ve skutečnosti pracuje grafická karta, závisí na ovladači a Vulkan slouží jen jako rozhraní pro požadavky na grafickou kartu.

Příkazy
Pomocí Vulkan API můžeme grafické kartě posílat požadavky. Tyto příkazy balíme jako balíčky do tzv. *příkazových bufferů*, aneb *command bufferů*. Např. v DirectX 12 se balíčky nazývají *command listy*, což je trochu výstižnější název. Dále jen *příkazové balíčky* nebo jen *balíčky*, bude-li *příkazové* zřejmé z kontextu.

Recording
Proces, kdy do balíčku postupně přidáváme příkazy, se nazývá *recording*, neboli *nahrávání*. Nejprve oznámíme začátek nahrávání konkrétního balíčku pomocí funkce *vkBeginCommandBuffer*. Poté přidáváme příkazy pomocí. Nakonec vše ukončíme pomocí `vkEndCommandBuffer`.

Pro představu:

```
vkBeginCommandBuffer(commandBuffer); // začni nahrávat

vkCmdCopyBufferToImage(commandBuffer, ...); // přidej příkaz: zkopíruj obsah nějakého
bufferu do obrázku

vkCmdDraw(commandBuffer, ...); // přidej příkaz: vykresli něco

vkEndCommandBuffer(commandBuffer); // ukonči nahrávání
```

Můžeme si všimnout, že volání, které přidá příkaz do balíčku, začíná `vkCmd`.

V předchozích API, jako OpenGL nebo DirectX 11, se příkazy takto neshlukovali. Bylo v režii ovladače zkusit příkazy optimalizovat a poslat do fronty. Nevýhodou bylo, že se muselo vše optimalizovat znovu a znovu každý snímek. Balíčky na druhou stranu můžeme nahrát jednou a posílat je už optimalizované vícekrát. Další výhodou je, že je možné nahrávat více balíčku současně, např. z různých vláken, a tím ještě více program zrychlit.

Posílání (Submit)

Fronta

Když chceme, aby se příkazy z balíčku vykonali, pošleme je do tzv. *fronty*. Protože CPU a GPU nejsou synchronizované, nemůžeme začít výpočet okamžitě. Proto se používá *fronta*, kterou grafická karta postupně vyprazdňuje.

Každá fronta je určité *rodiny*, kde každá *rodina* umí vykonávat určitou sadu příkazů. Obecně jsou 3 sady:

- Grafická: Především vykreslování, ale jestliže *rodina* umí tuto sadu, umí i *transfer* sadu
- Transfer: Pro přesun dat na grafické kartě.
- Compute:

Fronty ale nejsou tzv. *thread safe*, tedy nejsou připravené, aby se do nich nahrávalo z více vláken na CPU zároveň. Proto se jich vytváří více, většinou jedna pro každé vlákno, které k posílání budeme používat.

Synchronizace

Je zaručeno, že grafická karta začne balíčky a příkazy v něm vykonávat ve stejném pořadí, jako do fronty přišli. Není ale zaručeno, že skončí ve stejném pořadí. To je problém, protože často se používá vícero iterací, než vznikne výsledný obrázek. Je tedy třeba, aby každá iterace proběhla ve správném pořadí.

Command bufferu

Můžeme synchronizovat command buffery pomocí semaforů.

Frontu

Příkazy ve frontě můžeme synchronizovat pomocí bariér.

GPU to CPU

Pro synchronizaci mezi grafickou kartou a procesorem je tzv. fence, neboli plot.