Swapchain recreate

Swapchain

Swapchain is an object in Vulkan that facilitates image presentation onto surfaces. Surfaces are an abstraction over regions of the physical display, intended mainly for windowing systems and compositors. A swapchain is created for a combination of a surface and a device.

Requirement for our Swapchain object are:

1. Only one swapchain can exist for one surface.
2. Allow user to retrieve the surface when the swapchain is no longer in use.
3. Allow user to recreate the swapchain, transferring the ownership of the surface to the new instance, retiring the old swapchain.
4. Keep retired swapchain alive until all its acquired images are not longer in use.

Satisfying all three conditions as they are is not trivial, mainly because the the first two conditions lead to the requirement of dropping the swapchain once the surface is moved out of it, however, the fourth condition requires us to keep it alive. This can also create problems where for some reason the retired swapchain outlives the active one. In such cases, the surface can happen to be dropped before the retired swapchain, which is incorrect.

To satisfy all 4 conditions, we first have to rewrite them into terms that can be expressed in the language.

1. The creation of a swapchain requires full ownership of the surface, thus our constructor has to take surface by value.
2. The swapchain has to have a method that consumes the swapchain and returns the surface by value.
3. The new, recreated swapchain has to take the old swapchain by value and extract the surface from it using method from 2.
4. The swapchain has to be reference counted to outlive all its images.

Now it is much clearer why these requirements are hard to satisfy - 4. requires that the swapchain reference counted and its lifetime is guarded dynamically, however, 2. and 3. require for the lifetime of the swapchain to end immediatelly rather than sometime in the future. We need to rewrite the requirements to work with reference counting.

Adapting 2. is implementationally trivial. We must rely on the user to first drop all outstanding shared pointers except for one and then use that one to retrieve the swapchain back as an owned value.

Adapting 3. however, is much harder to implement as we can't expect the user to wait until all outstanding operations on the current swapchain are done until creating a new one since that would limit the functionality too much. Instead, we need to make sure that the surface is alive for the longer of the two lifetimes. This is exactly what reference counting does. By reference counting the surface inside a swapchain but still requiring an owned value for swapchain creation, we can make sure that no two active swapchains are ever created for one surface while still leaving the possibility of retrieving the surface after all but one of the shared pointers are dropped.

The resulting API thus looks like this:

```rust
pub struct Swapchain {
    surface: Vrc<Surface>,
    // Other fields
}
impl Swapchain {
    pub fn new(
        surface: Surface,
        // Other parameters
    ) -> Vrc<Self> {
        Vrc::new(
            Swapchain {
                surface: Vrc::new(surface),
                // Other fields
            }
        )
    }

    pub fn recreate(
        self: &Vrc<Self>,
        // Other parameters
    ) -> Vrc<Self> {
        Vrc::new(
            Swapchain {
                surface: self.surface.clone(),
                // Other fields
            }
        )
    }

    pub fn surface(&self) -> &Vrc<Self> {
        &self.surface
    }
}
```

This satisfies all the rules:

1. We cannot retrieve the surface back from the swapchain without destroying the shared pointer, which dynamically ensures there are no other instances.
2. The swapchain returns a reference to the reference counted surface, which can be destroyed to gain the surface after dropping all swapchains and swapchain images in the same way as above.
3. Both the new and the old swapchain contain a reference to the surface and thus will keep it alive for as long as is needed.
4. Swapchain is reference counted and can be kept alive by the images.