NVIDIA WaveWorks 1.6

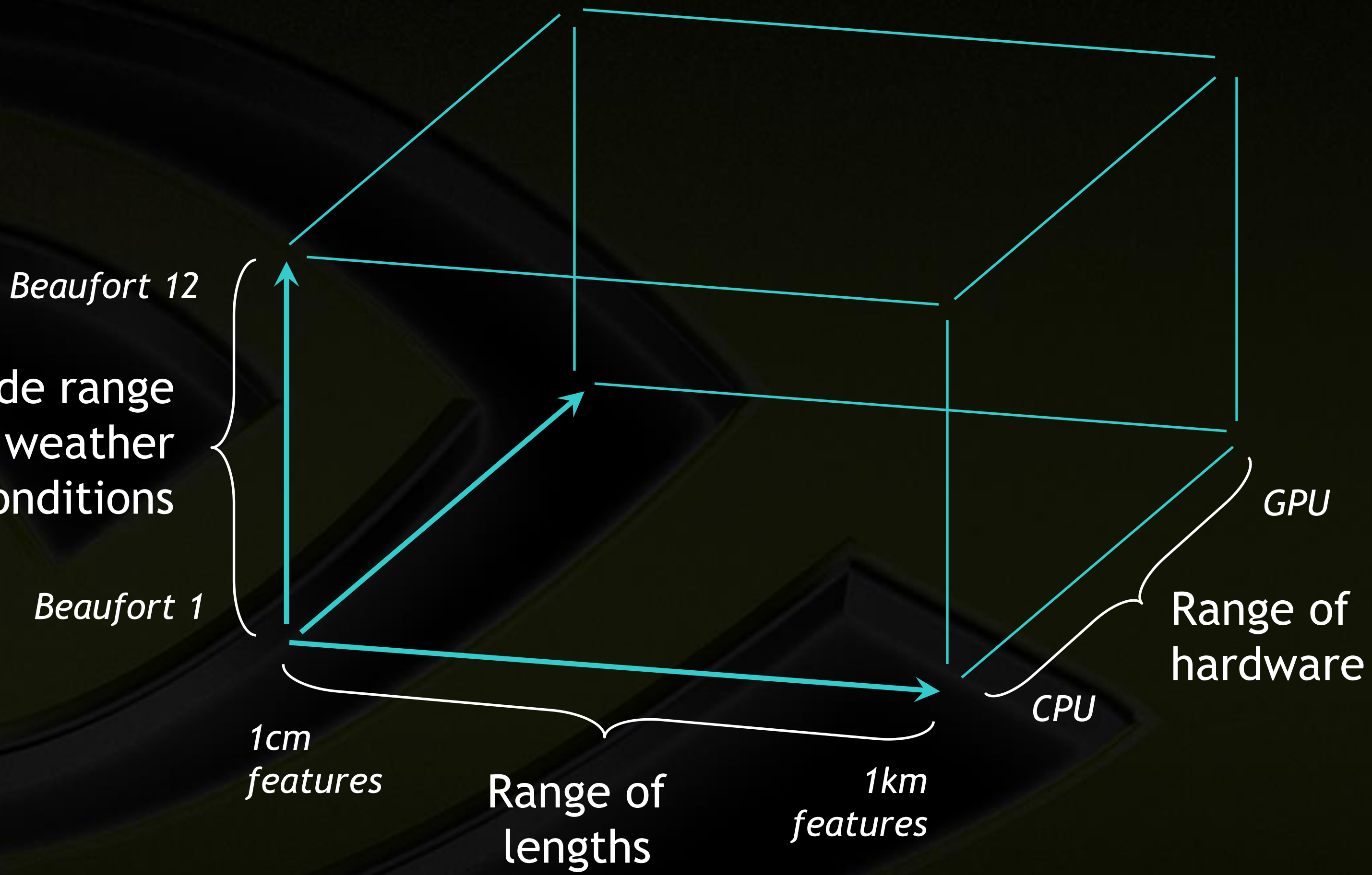
NVIDIA

NVIDIA WaveWorks

...a library for simulating wind-driven waves on large bodies of water, in real time, using GPU acceleration

(<u>BUT</u>: the application still defines the look)

Mission space



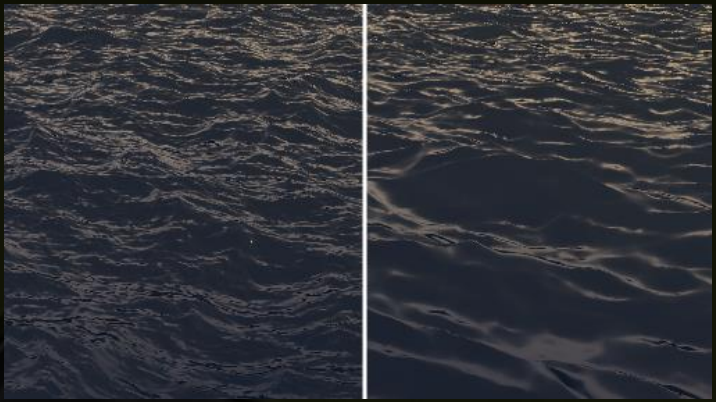
Detour: The Beaufort scale

- An empirical observation-based wind speed scale
- Devised 1805 by Francis Beaufort, British Navy
- Originally 0 to 12
  - 0 = Flat
  - 6 = Long waves begin to form. White foam crests are very frequent. Some airborne spray is present.
  - 12 = Huge waves. Sea is completely white with foam and spray. Air is filled with driving spray, greatly reducing visibility.
- Modern extensions to 13 and more

The length range challenge

- **Need both uniqueness and fine detail for a simulation to look good**

UNIQUENESS: minimize objectionable repeating patterns over large areas



FINE DETAIL: accurately portray near-camera short-wavelength features

The length range challenge

- **WaveWorks supports a length range of 64,000 : 1 –**

  64,000 represents the uniqueness dimension within which the simulation produces non-repeating results

  1 is the finest spacing of height samples seen near-camera

  e.g. 1.5cm detail vs. 1km uniqueness

  Handles 1km-order wavelengths

  - Beaufort 12 relevance

The algo

Tessendorf’s algo, based on Phillips spectrum

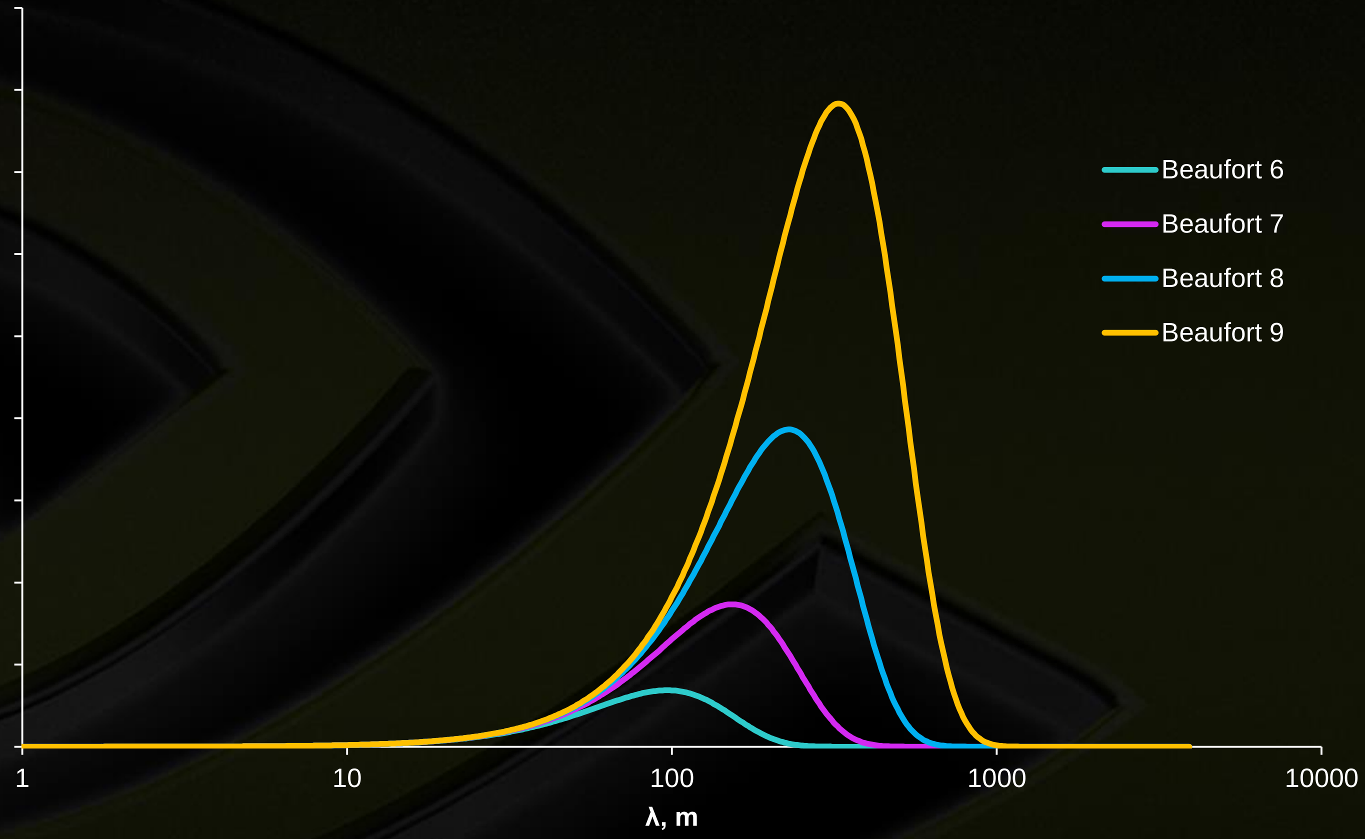
Waterworld & Titanic - ’97 : Life of Pi more recently

Simulation step runs in <2ms on GTX680 at max setting

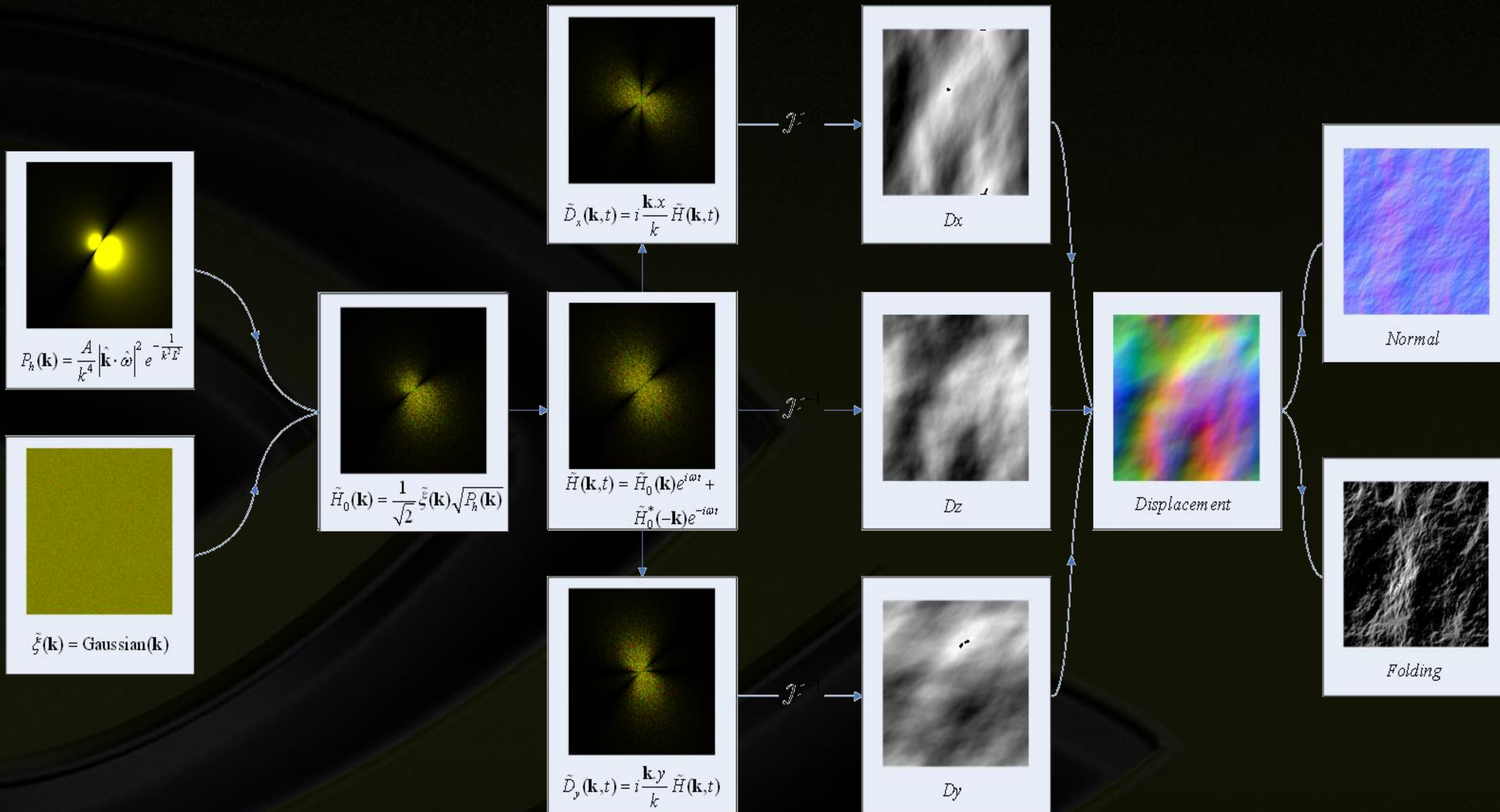
Explainer: the Phillips spectrum

- Predicted by Phillips in 1950s, working from first principles of Fluid Mechanics
- Subsequently validated by experimental oceanographic data in 1960s
- Models a ‘fully developed’ sea state
- So a great choice for ‘default’ spectrum
  - NB: rest of algo is spectrum-agnostic

Explainer: the Phillips spectrum



Simulation pipeline



Per-params-change (CUDA/CPU)

Per-frame (CUDA/CPU)

Per-frame (PS)

Overview of Version 1.5

- **Multi-res simulation for 64,000 : 1 length range**
- **Geometry LODing**
  - **Coarse level: quad tree**
  - **Fine level, D3D9/10: geo-morphing**
  - **Fine level, D3D11: tessellation**
- **Lighting/shading remains under full application control**
- **Simulation and rendering are controlled via a simple C API**
  - **Simulation results accessed via HLSL shader fragments that ship with the lib**

Supported configurations

	CPU sim	*GPU sim*
Windows/D3D9	✓	
Windows/D3D9Ex	✓	✓
Windows/D3D10	✓	✓
Windows/D3D11	✓	✓
Windows/GL	✓	✓
Windows/sim-only	✓	✓
Linux/sim-only	✓	✓
PS4	✓	
Xbone	✓	✓

How to exploit

- The obvious use-case: render a good-looking sea
- Also: as prime mover for 2ry/3ry effects
  - For vessel physics
  - For vessel spray/foam production
    - And from there, audio for spray/foam
  - Conceptually: a big high-quality animated noise field

Ease-of-use, ease-of-integration

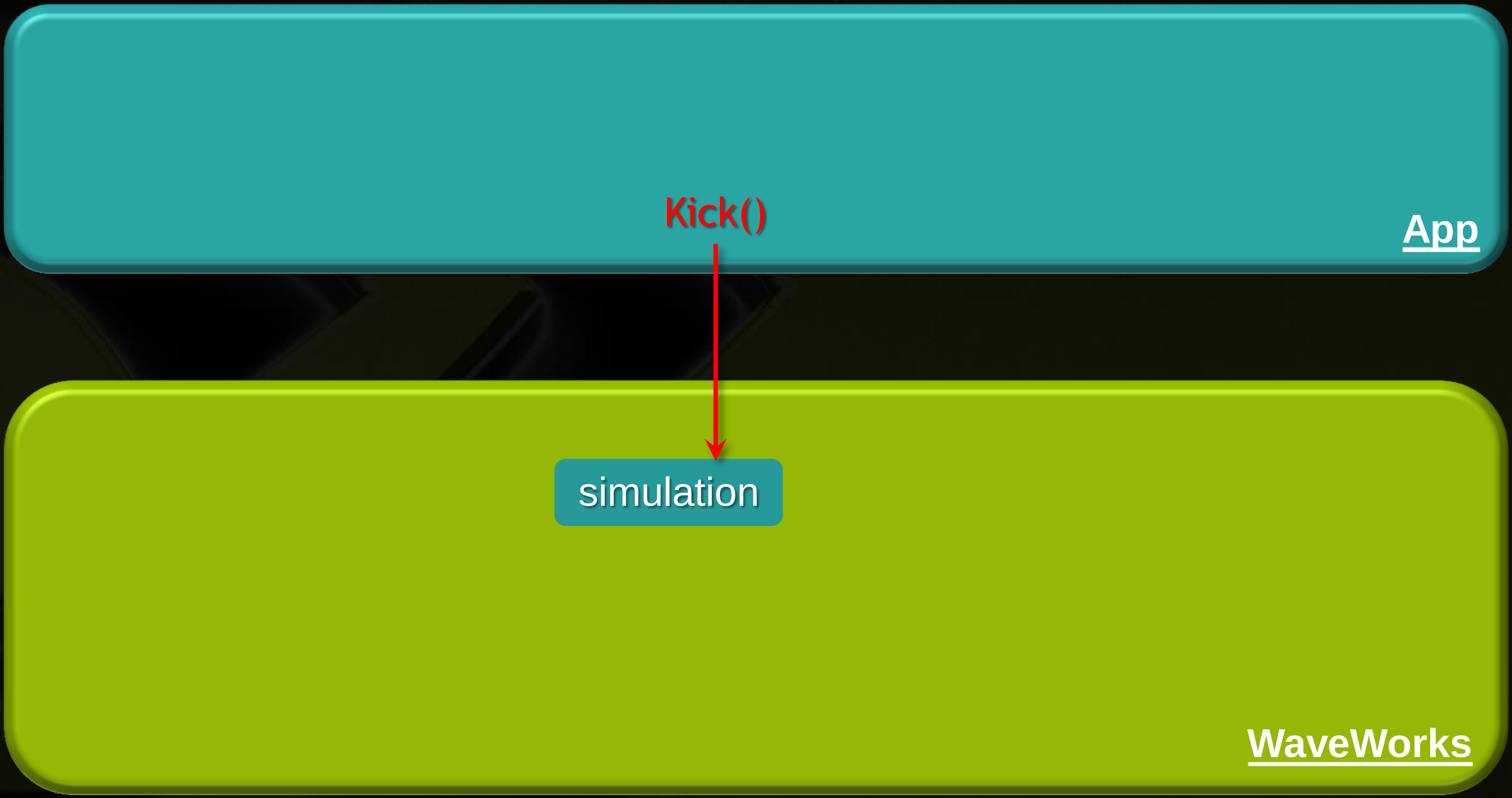
- 1.5 can be integrated with just 23 API entrypoints
- Typically 2 days to 1 week for initial integration
- Optional save/restore of D3D device state across API calls
- Also supports selective reading of height data back to host (‘readback’)
  - E.g. to feed into physics simulation of water-borne objects, like boats

API sample - simulation

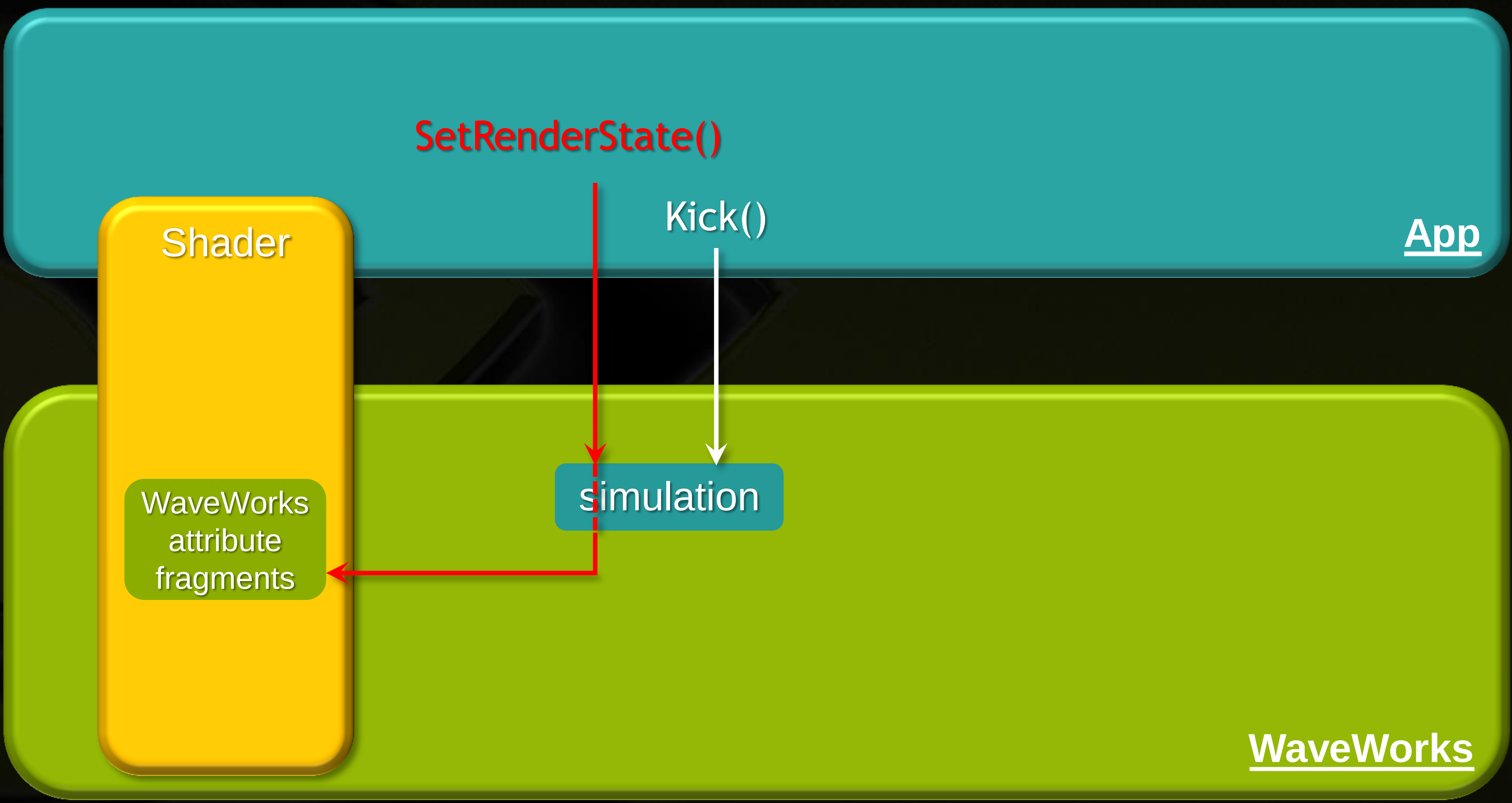
```
// Initialisation
GFSDK_Waveworks_Simulation_CreateD3D11(
    const GFSDK_Waveworks_Simulation_Settings& settings,
    const GFSDK_Waveworks_Simulation_Params& params,
    ID3D11Device* pD3DDevice,
    GFSDK_Waveworks_SimulationHandle* pResult
);
// Per-frame updates
GFSDK_Waveworks_Simulation_SetTime(
    GFSDK_Waveworks_SimulationHandle hSim,
    gfsdk_f64 dAppTime
);
GFSDK_Waveworks_Simulation_KickD3D11(
    GFSDK_Waveworks_SimulationHandle hSim,
    ID3D11DeviceContext* pDC,
    GFSDK_Waveworks_SavestateHandle hSavestate
);
// Setting state for rendering
GFSDK_Waveworks_Simulation_SetRenderStateD3D11(
    GFSDK_Waveworks_SimulationHandle hSim,
    ID3D11DeviceContext* pDC,
    const gfsdk_float4x4& matView,
    const gfsdk_U32* pShaderInputRegisterMappings,
    GFSDK_Waveworks_SavestateHandle hSavestate
);
```

Setting up a simulation (one-time)

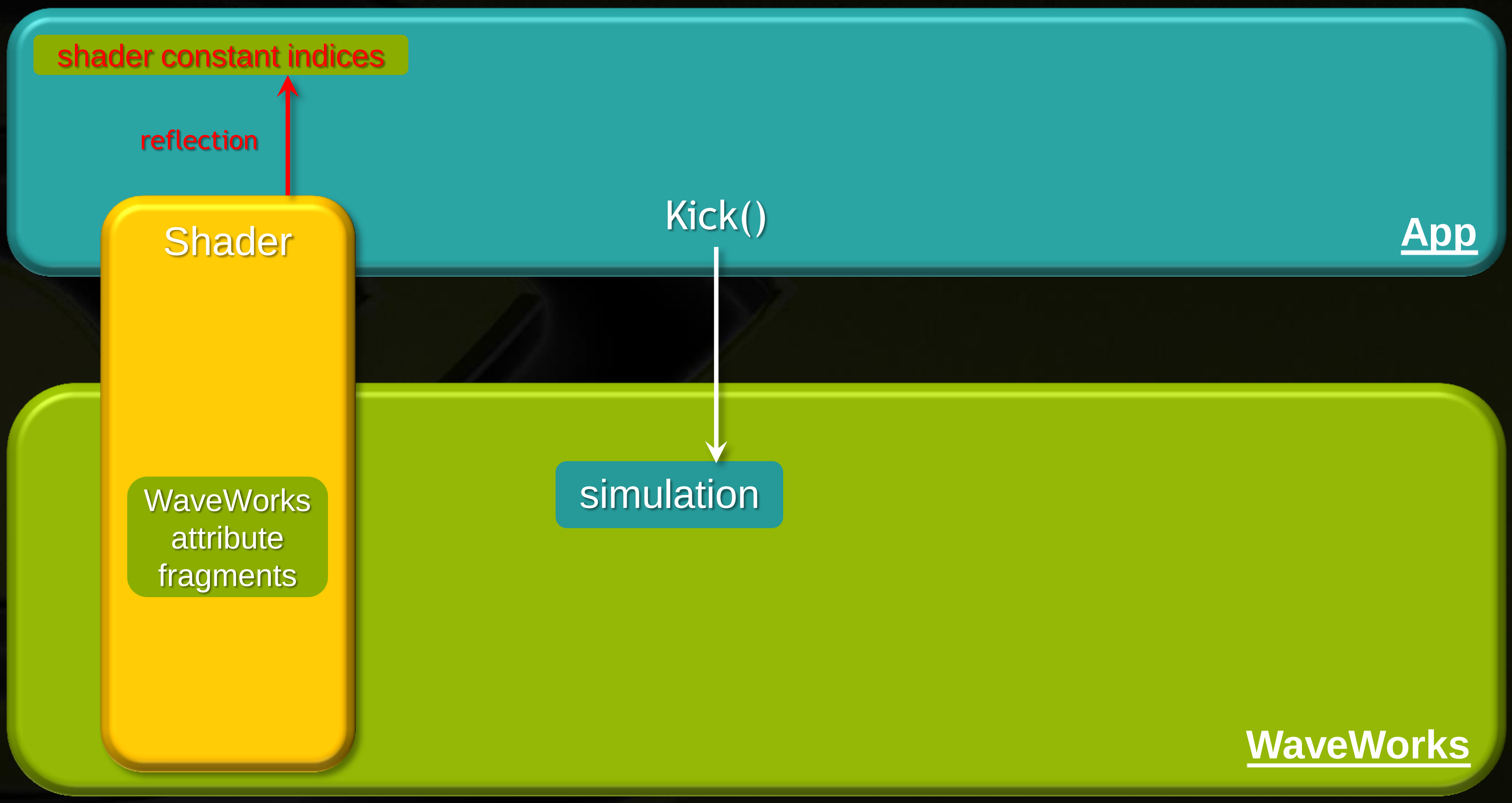
Pumping the sim (per frame)



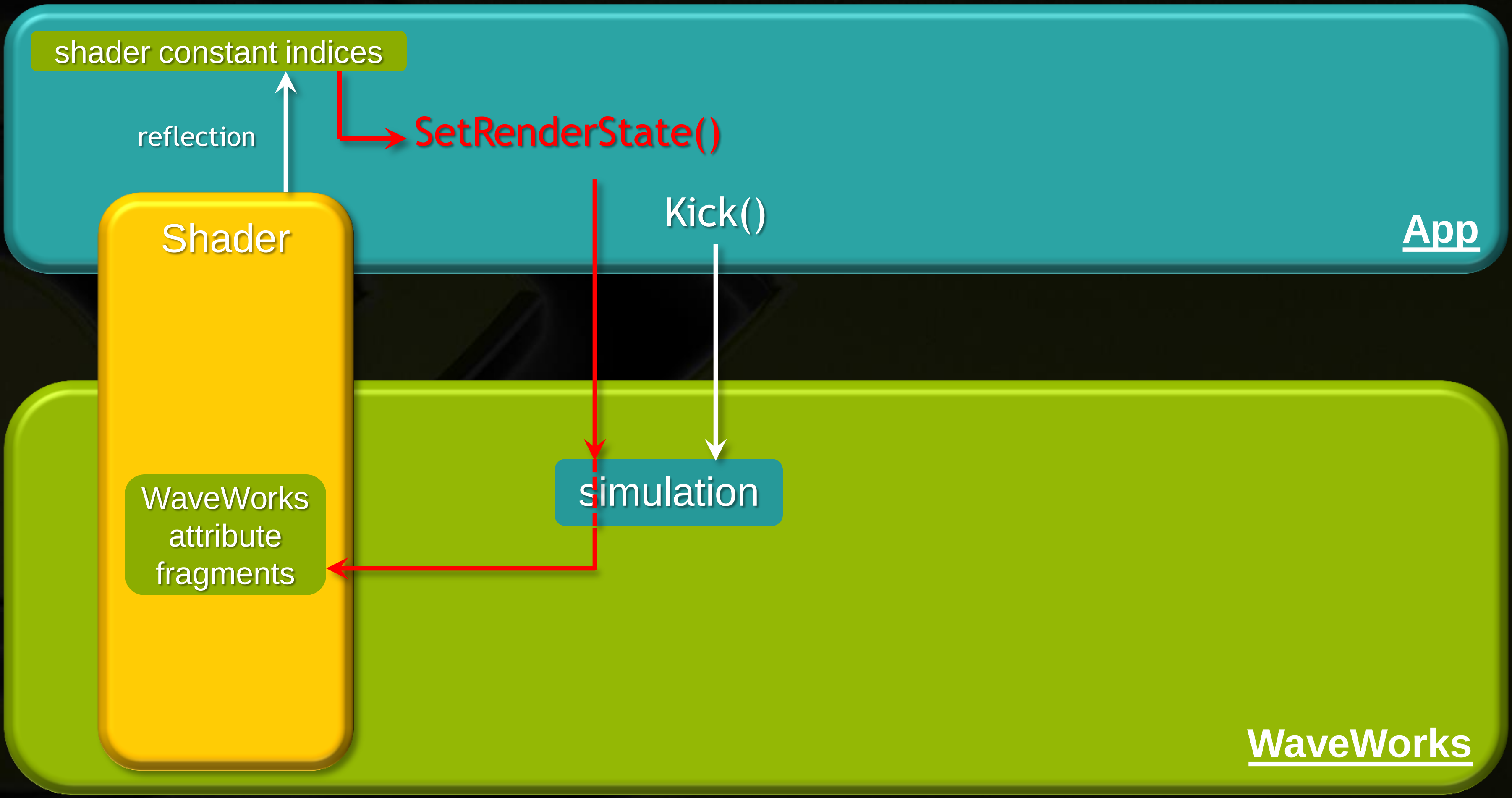
Rendering the results (per frame)



Shader constants setup (one-time)



Rendering the results (per frame)

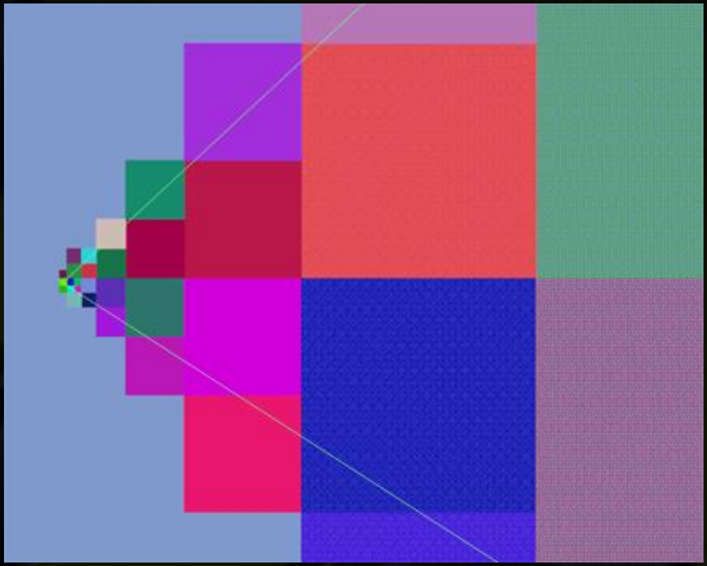


Shader integration

- Shader hookup is likely trickiest aspect of integration
- Sample app: ships as source within distro
- Includes an example using -
  - Named constants
  - FX file format
  - Shader reflection to get constant offsets from compiled FX's
- Good way to follow the workings
- (But we also support fixed reg allocation schemes)

Quad-tree Drawing

- Quad-tree used for frustum culling and mesh LOD



- Quad-tree tiling calculated in world space
  - Ensures that mesh is stable w.r.t camera rotation

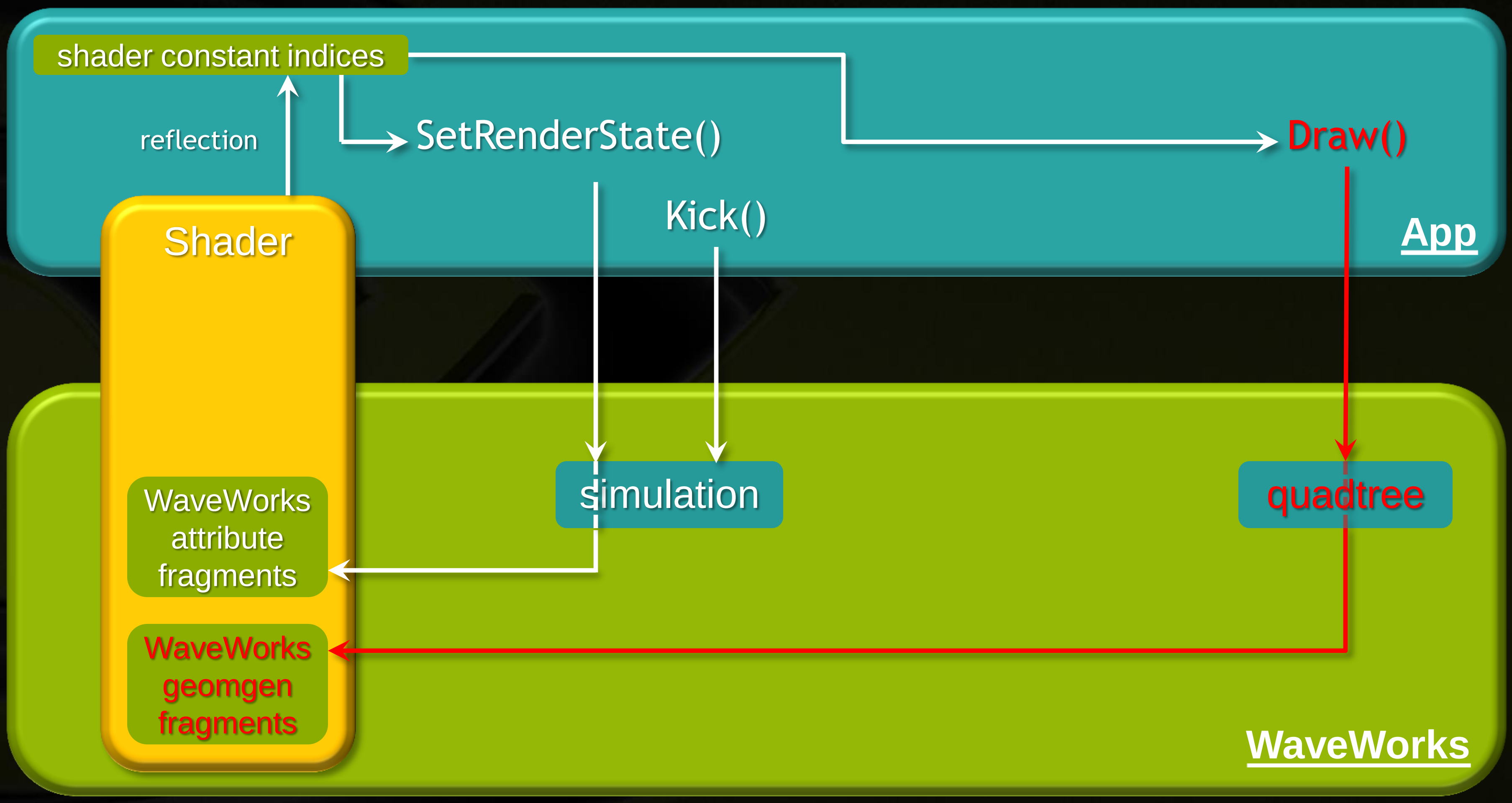
Quad-tree Drawing (cont)

```
// Initialisation
GFSDK_Waveworks_Quadtree_CreateD3D11(
    const GFSDK_Waveworks_Quadtree_Params& params,
    ID3D11Device* pD3DDevice,
    GFSDK_Waveworks_QuadtreeHandle* pResult
);

// Drawing
GFSDK_Waveworks_Quadtree_Draw(
    GFSDK_Waveworks_QuadtreeHandle hQuadtree,
    ID3D11DeviceContext* pDC,
    const gfsdk_float4x4& matView,
    const gfsdk_float4x4& matProj,
    const gfsdk_U32* pShaderInputRegisterMappings,
    GFSDK_Waveworks_SavestateHandle hSavestate
);
```

- DX11 path uses tessellation to add fine mesh detail
- DX9/10 uses geo-morphing for smooth triangulation
- Tiles can be individually enabled/disabled via the API to handle non-water (i.e. inland) areas

Quad-tree Drawing (per frame)



Artistic tuning knobs

```
struct GFSDK_WaveWorks_Simulation_Params
{
    // Simulation properties
    gfsdk_F32 wave_amplitude;
    gfsdk_float2 wind_dir;
    gfsdk_F32 wind_speed;        // accepts either meters/sec or Beaufort scale
                                 // value, depending on UseBeaufortScale in
                                 // GFSDK_WaveWorks_Simulation_Settings struct
    gfsdk_F32 wind_dependency;
    gfsdk_F32 choppy_scale;
    gfsdk_F32 small_wave_fraction;

    // The overall time scale for the simulation (FFT)
    gfsdk_F32 time_scale;

    // the factor characterizing critical wave amplitude/shape/energy to start generating foam
    gfsdk_F32 foam_generation_threshold;
    // the amount of foam generated in such areas on each simulation step
    gfsdk_F32 foam_generation_amount;
    // the speed of foam spatial dissipation
    gfsdk_F32 foam_dissipation_speed;
    // the speed of foam dissipation over time
    gfsdk_F32 foam_falloff_speed;
};
```

Indicative performance

Detail level setting	Hardware	Elapsed time for simulation step
Normal	Core i7-2600K	1.1ms*
High	GTX 680	1.0ms
Extreme	GTX 680	1.4ms

*(using 4 of 8 logical cores)

Coming soon: Version 1.7