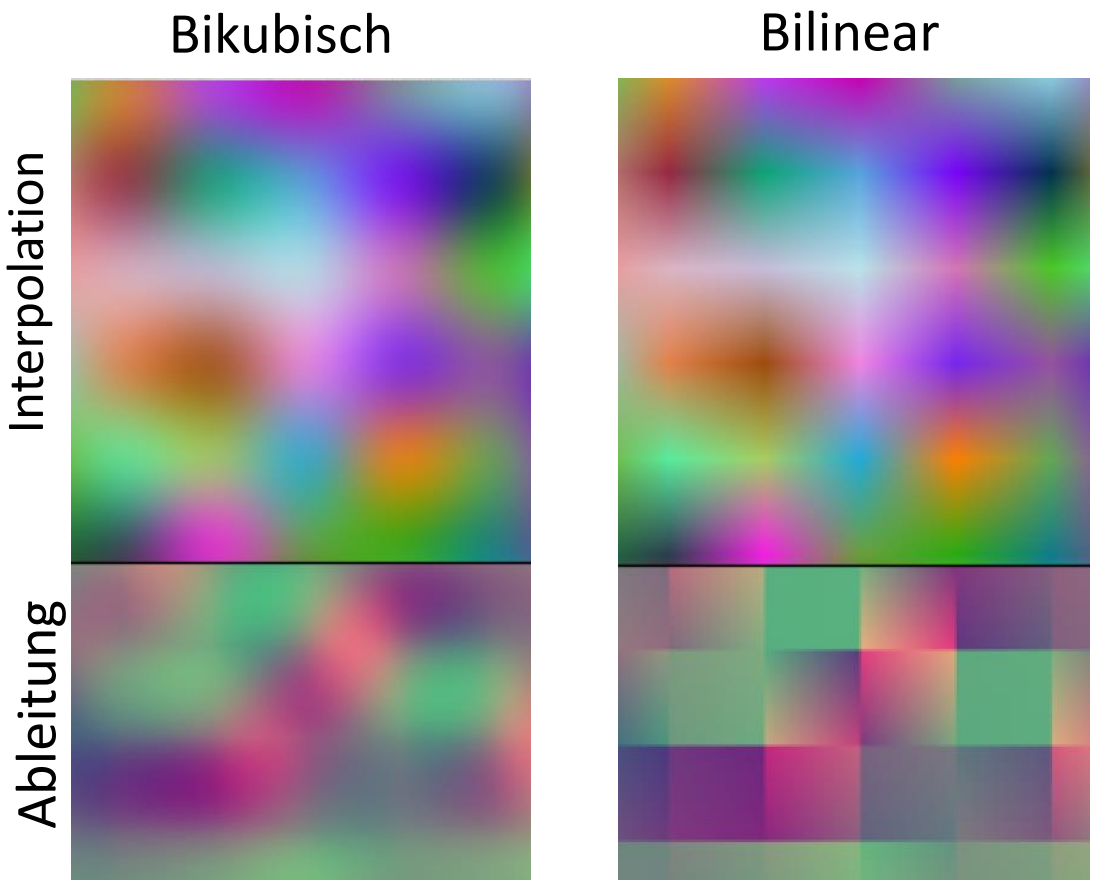
# Übung: Interaktive Computergrafik

# Heute

- Besprechung des dritten Übungsblatts (Variance Shadow Maps)
- Optimierung von Texturfilterung durch bilineare Interpolation
- Profiling

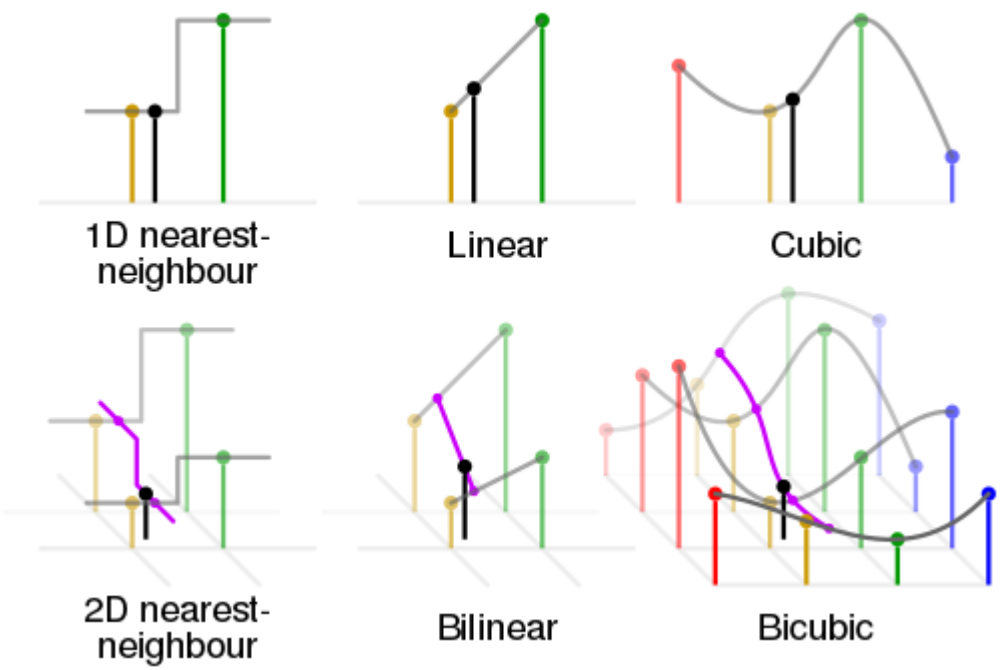
# Bicubic vs Bilinear




https://www.shadertoy.com/view/XsSXDy

# Bikubische Interpolation

- Naive Implementierung benötigt 16 Texturzugriffe



# Kubische Interpolation

- Für kubischen Filter muss folgende Summe ausgewertet werden:

$$w_0(x) \cdot f_{i-1} + w_1(x) \cdot f_i + w_2(x) \cdot f_{i+1} + w_3(x) \cdot f_{i+2}$$

- Idee: benutze lineare Interpolation die in Hardware unterstützt wird

$$f_x = (1 - \alpha) \cdot f_i + \alpha \cdot f_{i+1} \qquad i = \lfloor x \rfloor \quad \alpha = x - i$$

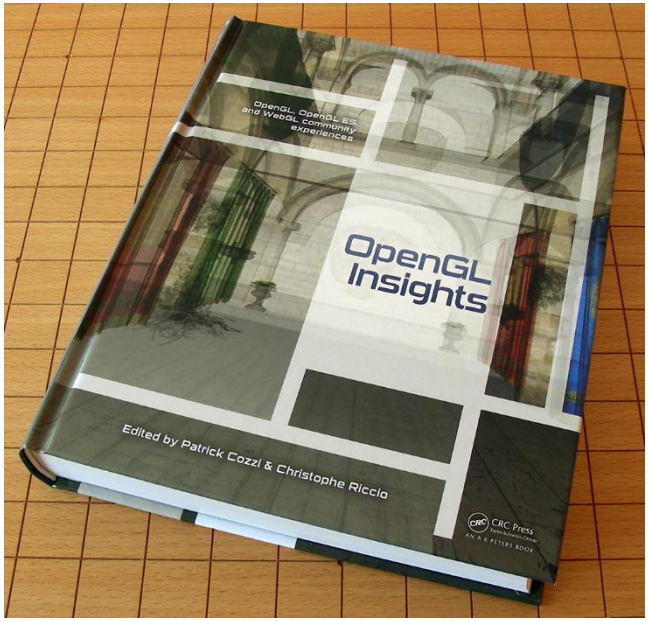
- Ersetze $a \cdot f_i + b \cdot f_{i+1}$ durch $(a + b) \cdot f_{i+b/(a+b)}$
  - Verschiebung von Texturkoordinate gibt relative Gewichtung

- Die Summe kann somit durch zwei Texturzugriffe ausgewertet werden
  - Analog in 2D: 4 Texturzugriffe statt 16
- https://www.vrvis.at/publications/pdfs/PB-VRVis-2005-012.pdf

# Profiling

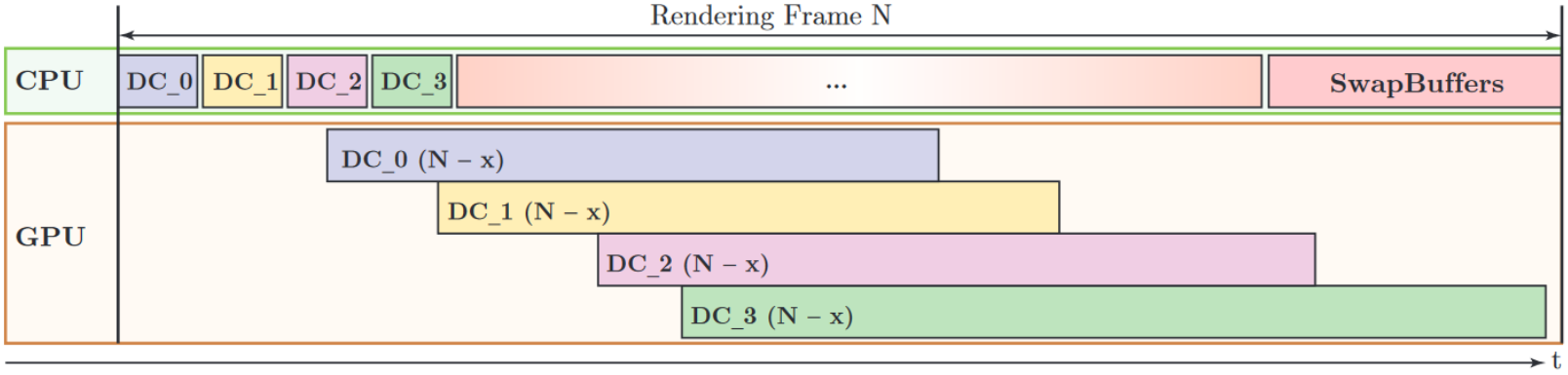
- Notwendig zur Messung und Optimierung von Rendering-Performance

- Nachfolgende Folien basieren auf Material aus OpenGL Insights



# Profiling

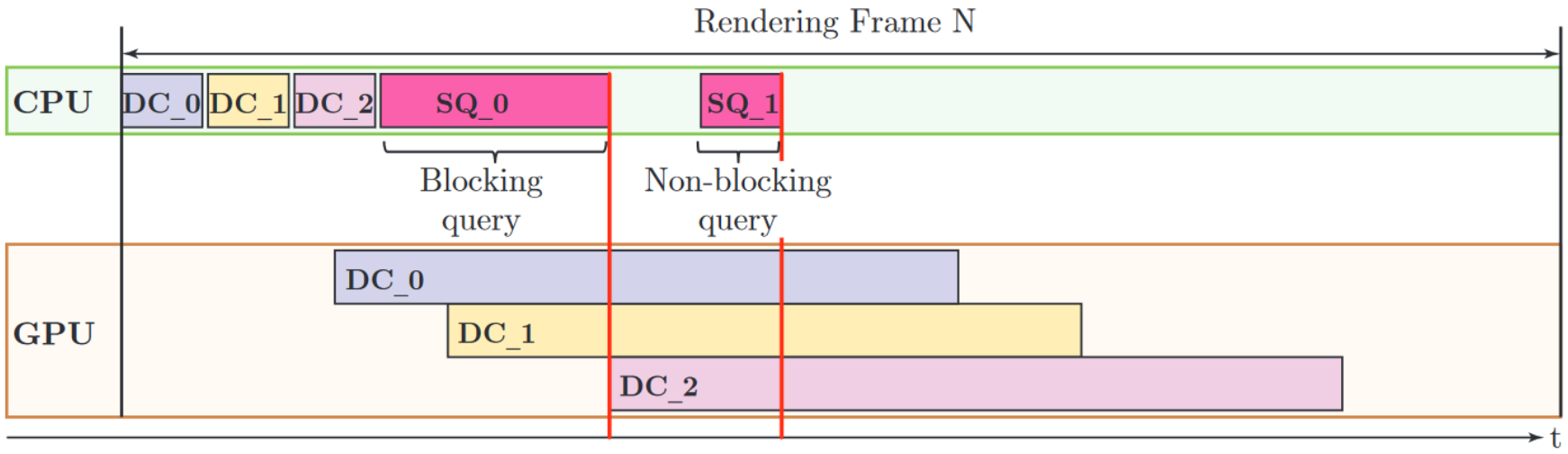
- CPU erzeugt draw-calls die durch Command-Queue abgearbeitet warden
- GPU läuft asynchron zur CPU
  - Ausführungszeiten auf der GPU können nicht ohne weiteres mit CPU-Timern bestimmt werden



# Synchrone Timestamp Queries

```
// this variable will hold current time on the GPU
GLint64 time_stamp;
// retrieve the current time stamp,
// after all prior OpenGL commands reached the GPU
glGetInteger64v(GL_TIMESTAMP, &time_stamp);
```

# Synchrone Timestamp Queries

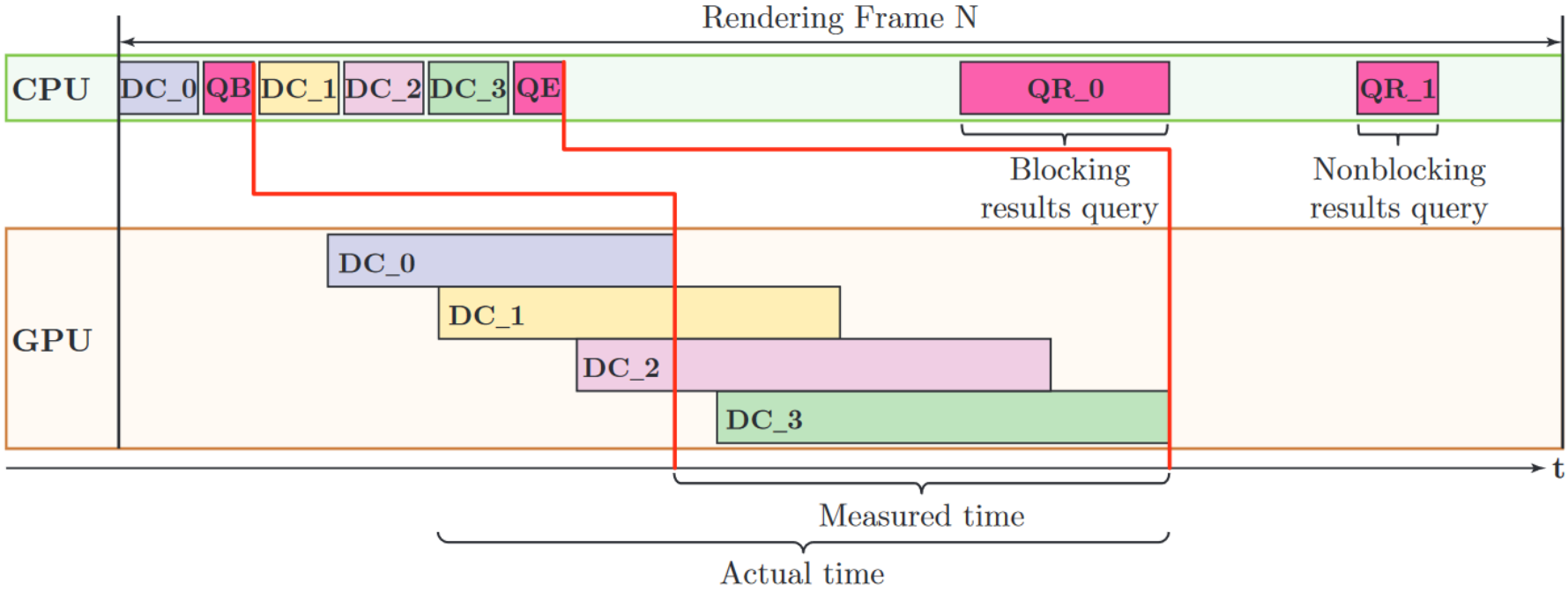



- CPU wird blockiert bis vorherige OpenGL Befehle die GPU erreicht haben
  - Timestamp für Beginn von Abarbeitung von DC_2 wird zurückgeliefert
  - SQ_1 blockiert nicht und gibt sofort einen Wert zurück

# Asynchrone Timer Queries

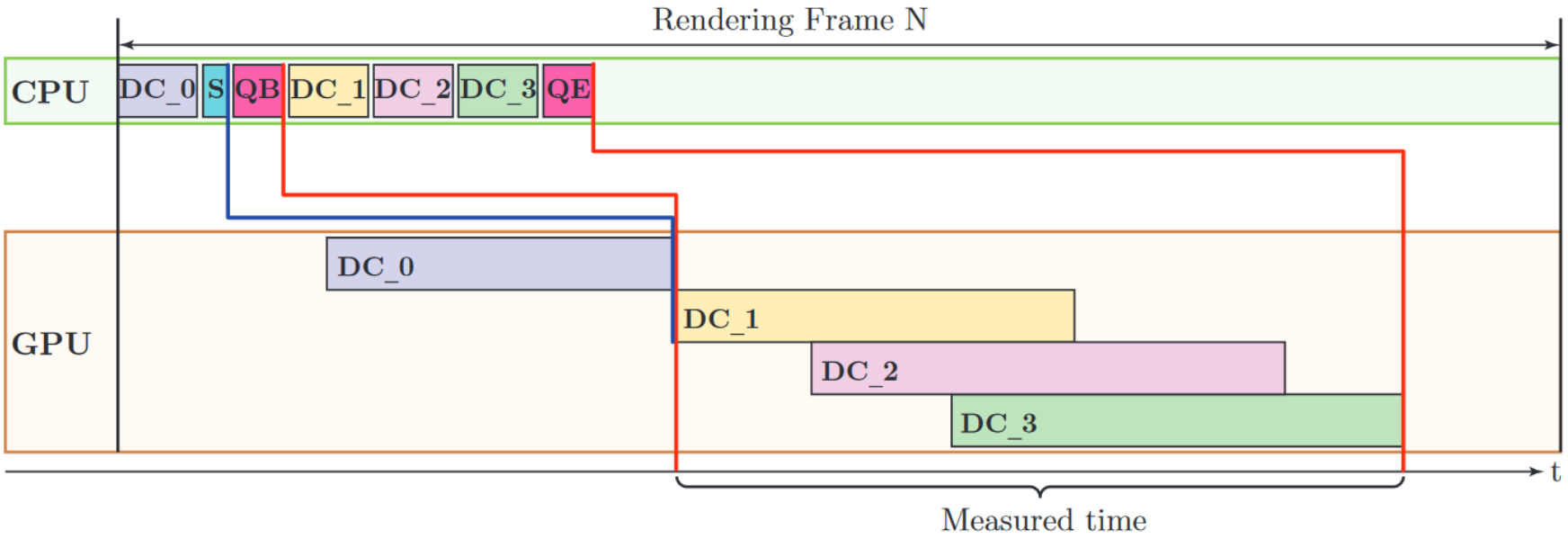
```
GLuint timer_query;
// generate query object
glGenQueries(1, &timer_query);
[...]
// start the timer query
glBeginQuery(GL_TIME_ELAPSED, timer_query);
// issue a sequence of OpenGL rendering commands
[...]
// stop the timer query
glEndQuery(GL_TIME_ELAPSED, timer_query);
[...]
// retrieve the query results, potentially stalling GPU execution
GLuint64 timer_result;
glGetQueryObjectui64v(timer_query, GL_QUERY_RESULT, &timer_result);

printf("GPU timing result: %f ms\n", double(timer_result) / 1e06);
```

# Asynchrone Timer Queries




- Draw calls können überlappend abgearbeitet werden

# Asynchrone Timer Queries




- Draw calls können überlappend abgearbeitet werden
  - Einfügen von expliziten Synchronisierungspunkten

# Asynchrone Timestamp Queries

```
GLuint timer_queries[3];
// generate three query objects
glGenQueries(3, &timer_queries);
[...]
// query time stamps around all draw routines
glQueryCounter(timer_queries[0], GL_TIMESTAMP);
draw_world();
glQueryCounter(timer_queries[1], GL_TIMESTAMP);
draw_models();
glQueryCounter(timer_queries[2], GL_TIMESTAMP);
[...]
// later in the program retrieve the query results
GLuint64 time_0, time_1, time_2;
glGetQueryObjectui64v(timer_queries[0], GL_QUERY_RESULT, &time_0);
glGetQueryObjectui64v(timer_queries[1], GL_QUERY_RESULT, &time_1);
glGetQueryObjectui64v(timer_queries[2], GL_QUERY_RESULT, &time_2);

printf("world draw time : %f ms\n", double(time_1 - time_0) / 1e06);
printf("models draw time: %f ms\n", double(time_2 - time_1) / 1e06);
```

- Eignen sich wenn man verschachtelte Blöcke messen will