The GAME Engineers

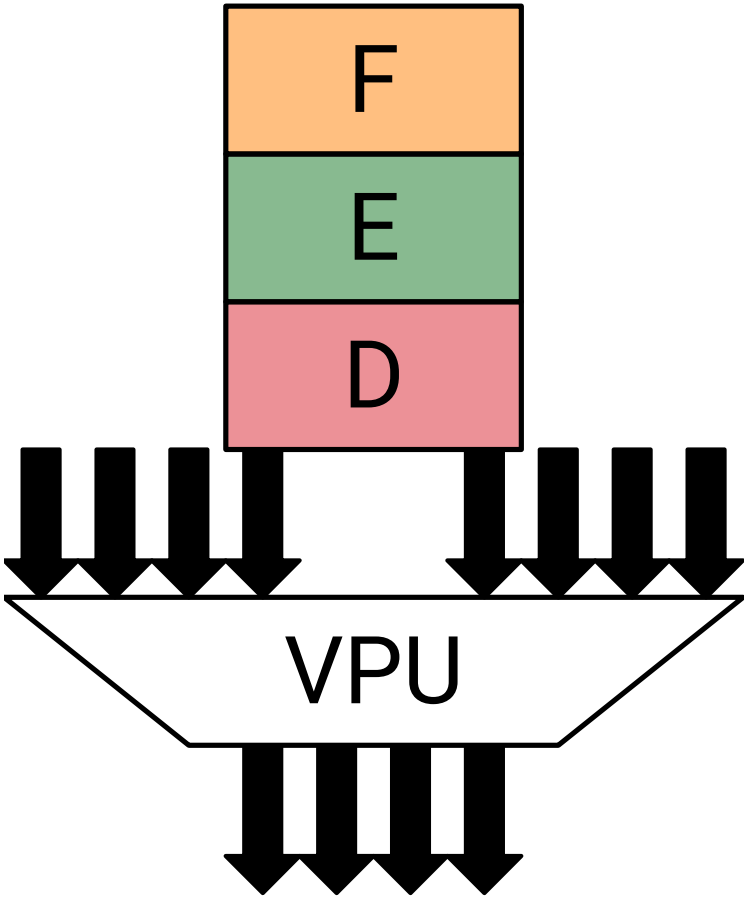
CG #7 – Grafik API und OpenGL

Inhalt

- Grafik API
  - GPU und Grafikkarte
  - API
  - OpenGL

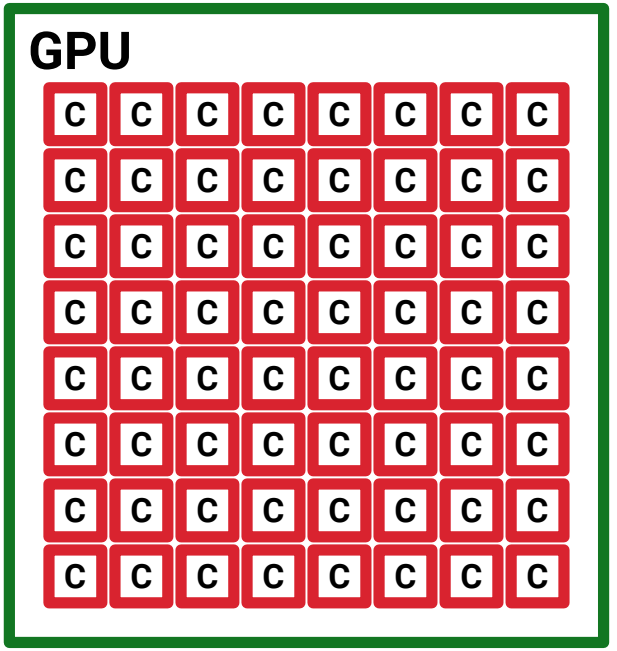
Grafik API – GPUs und Grafikkarte

- SIMD
  - z.B. Vectorprocessing
    - Viele Daten, auf die eine Operation angewendet werden soll



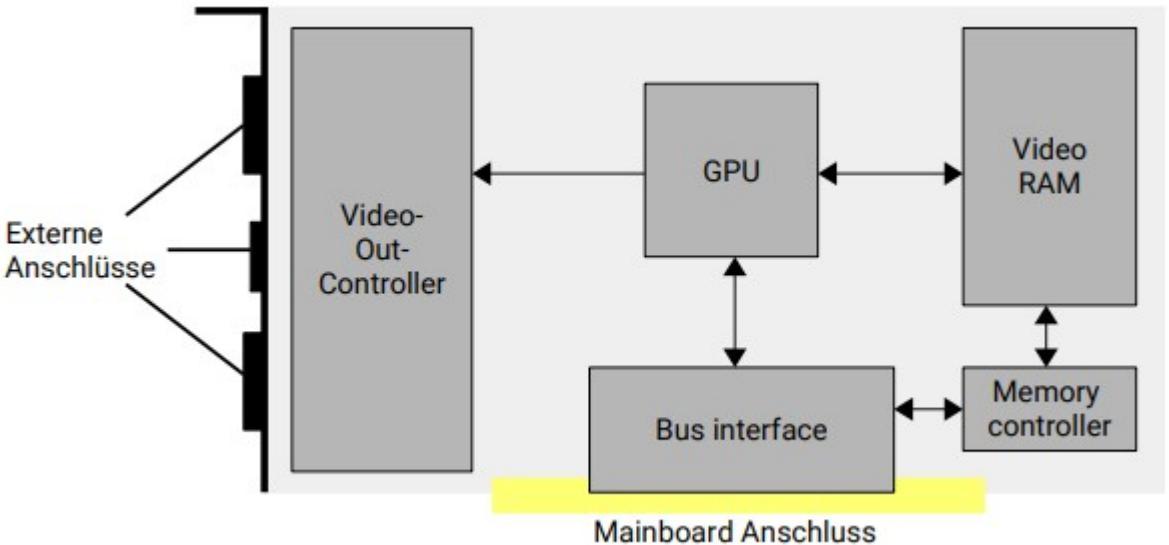
Grafik API – GPUs und Grafikkarte

- GPU
  - Cores werden Shader genannt
  - Nicht jeder Core tut das gleiche
    - Vertex Shader
    - Geometry Shader
    - Fragment Shader
    - …
  - Manche Shader(stages) können durch Shaderprogramme programmiert werden



Grafik API – GPUs und Grafikkarte

- Grafikkarte
  - Hardwaremodul mit
    - GPU
    - Eigenem Speicher (VRAM)
    - Anderen Teilen
  - Daten werden vom RAM über einen Bus zum VRAM transportiert
    - Geometriedaten
    - GPU Befehle, etc.

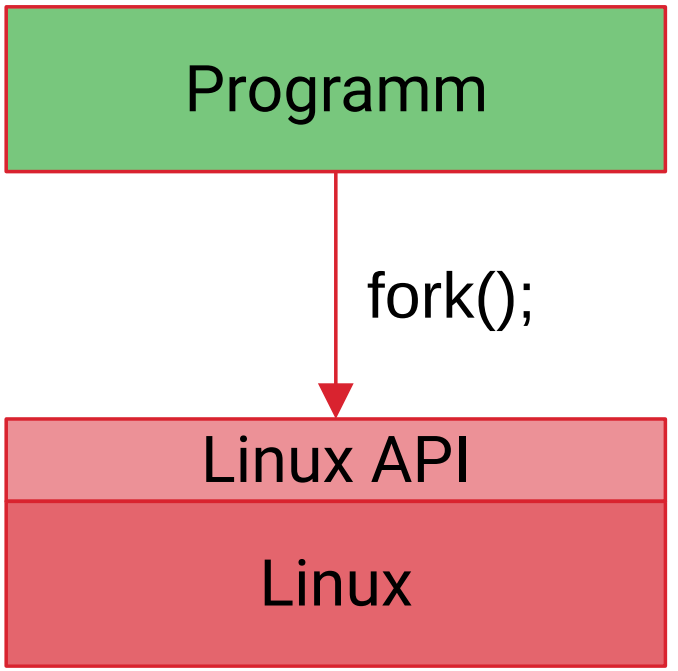


Doch wie sage ich als Programmierer, dass jetzt was auf die Graka soll bzw. was die GPU ausführen soll?

→ **Grafik API**

Grafik API – API

- Was ist eine API?
  - Schnittstelle, die ein Programmierer ansprechen kann
  - Bsp:
    - Zum OS
    - Zu Bibliotheken/Frameworks
    - Datenbankzugriff
    - …

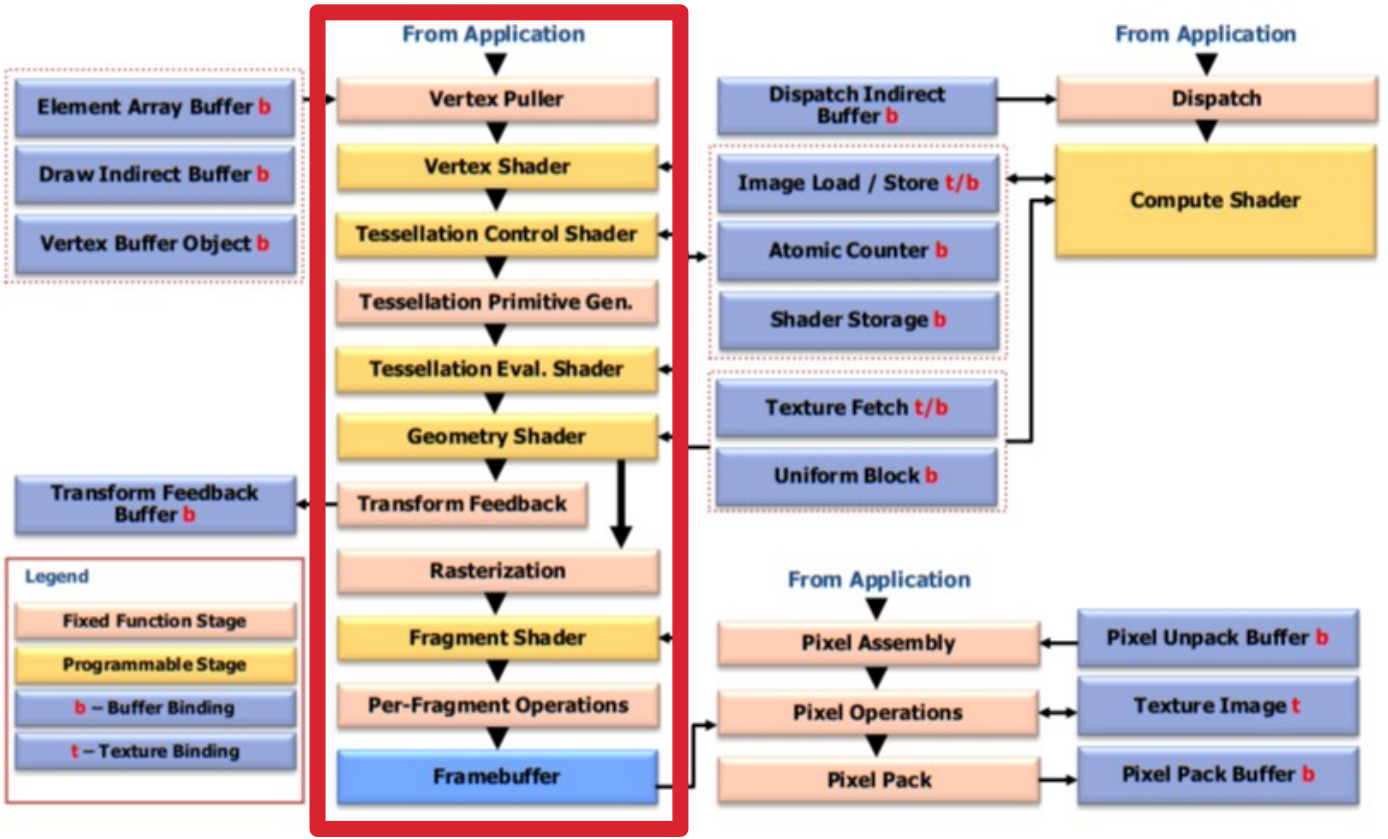


Eine Grafik API stellt also einfach nur ein (standardisierter) Zugriff auf die Grafikkarte und deren Funktionen zur Verfügung.

Grafik API – OpenGL

- OpenGL als Spezifikation schreibt vor, wie die API auszusehen hat und wie sie sich verhalten soll
  - Die GPU-Hersteller müssen dies dann implementieren, sodass ihre GPU das richtige tut
  - Wir als Programmierer können dann drauf zugreifen und uns ist es mehr oder weniger egal, welche GPU (AMD/Nvidia/etc.) benutzt wird, da die API ja vorschreibt, was wir in unserem Quellcode aufrufen können und was dann passieren soll

Grafik API – OpenGL



Grafik API – OpenGL

- State Machine
  - Vor jedem Renderdurchgang muss der Zustand wie gewünscht gesetzt/geändert werden

- GLSL als Shader-Programmiersprache

Grafik API – OpenGL

- Heute wird OpenGL mit Shadern verwendet
  - Früher gab es programmierbare Shader nicht
  - Shaderprogrammierung ist ein großer Teil und kommt wann anders
  - Wir verwenden erst mal die alte Variante zu zeichnen

- Farbe:

```
glColor3f(1, 1, 1);
```

- Draw Call:

```
glBegin(GL_POINTS);
    glVertex2f(0, 0);
glEnd();
```

```
glBegin(GL_LINES);
    glVertex2f(0, 0);
    glVertex2f(0, 1);
glEnd();
```

```
glBegin(GL_QUADS);
    glVertex2f(0, 0);
    glVertex2f(1, 0);
    glVertex2f(1, 1);
    glVertex2f(0, 1);
glEnd();
```

Grafik API – OpenGL

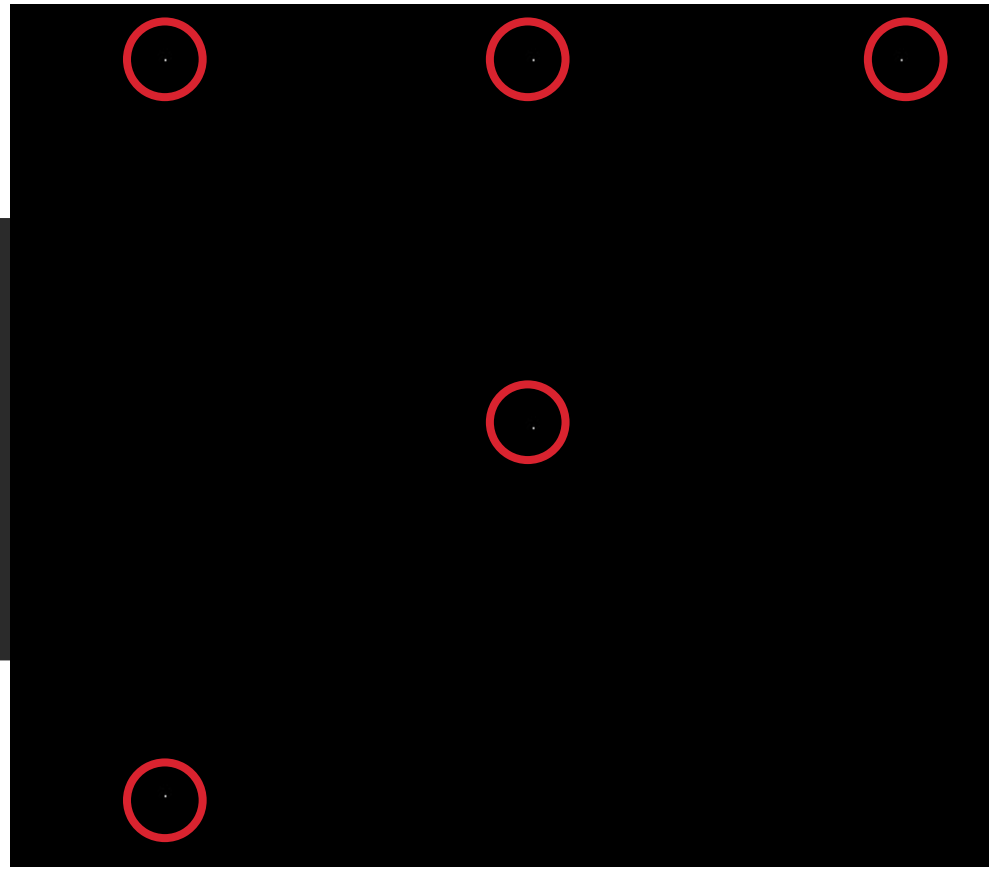
- glBegin(Mode)
  - Mode gibt an, wie die Punkte interpretiert werden sollen
    - GL_POINTS → Einzelne Punkte
    - GL_LINES → Je zwei Punkte werden durch Linie verbunden
    - GL_TRIANGLES → Je drei Punkte werden zu einem Dreieck verbunden (gefüllt)
    - GL_QUADS → Je vier Punkte werden zu einem
    - …
  - Je nach Mode ist das rendern langsamer oder schneller
    - Punkte brauchen länger als ein Dreieck (relativ)
    - Aber das fällt (für uns) kaum ins Gewicht

glBegin wird ab OpenGL nicht mehr unterstützt, da ab dieser Version alles Shaderbasiert ist. Mit der Compatibility-Version die wir ja nutzen geht das aber weiterhin.

Grafik API – OpenGL

- GL_POINTS
  - Min. 1 Punkt

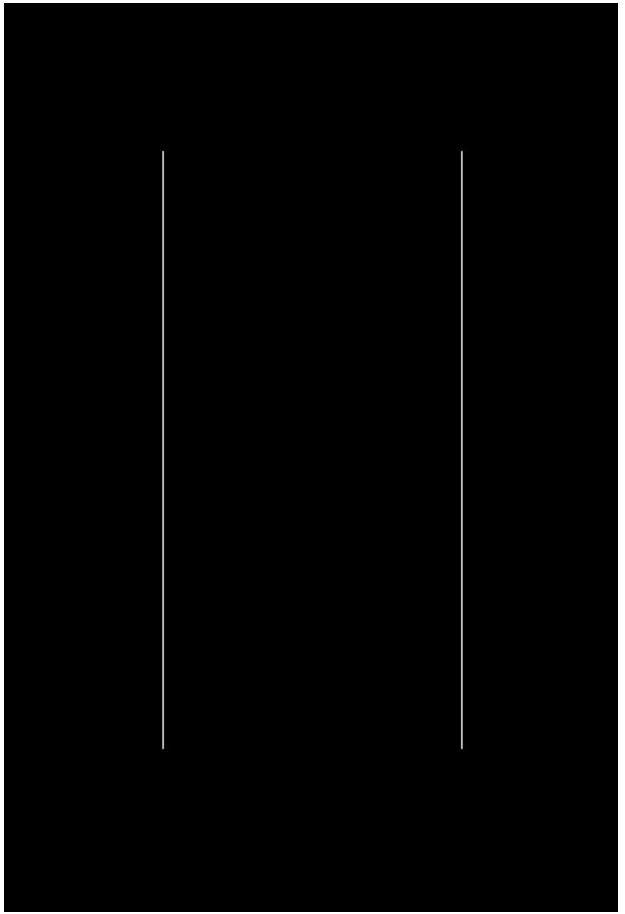
```
glBegin(GL_POINTS);
glVertex2f(0, 0);
glVertex2f(0, 0.5f);
glVertex2f(0.5f, 0.5f);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glEnd();
```



Grafik API – OpenGL

- GL_LINES
  - Min. 2 Punkt
  - Vielfaches von 2

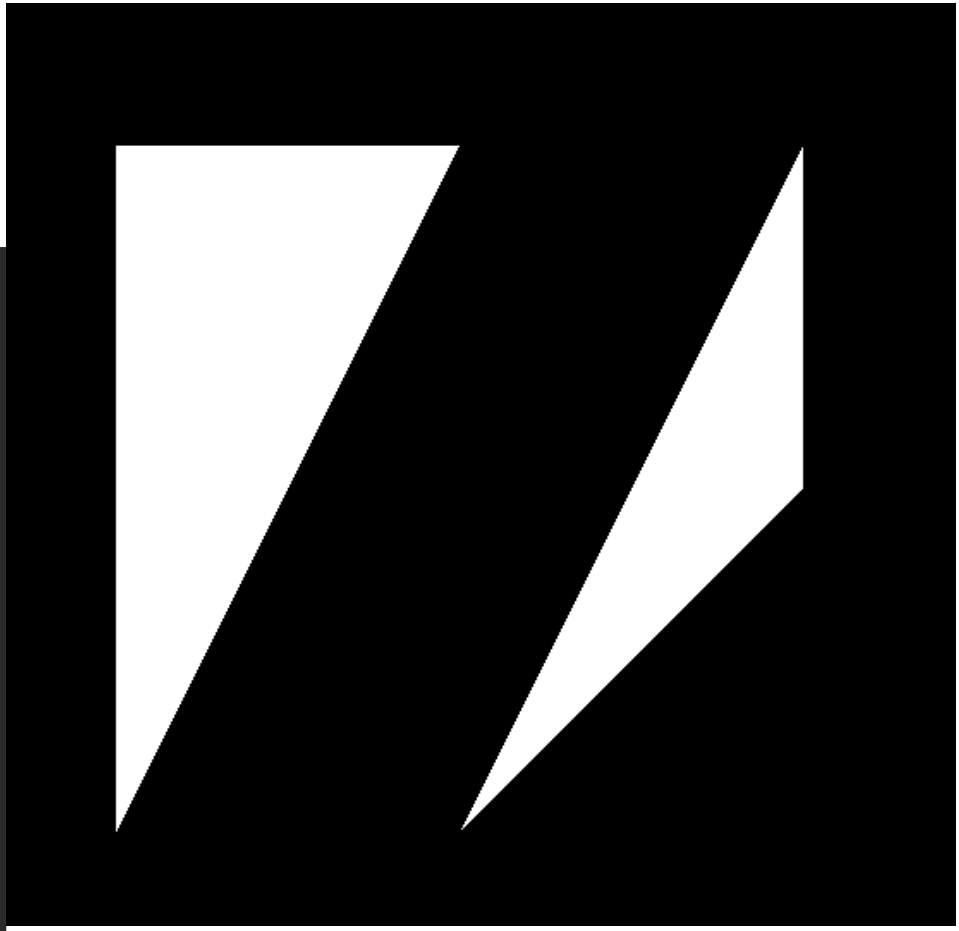
```
glBegin(GL_LINES);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);

glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);

glVertex2f(0.5f, 0.5f);
glEnd();
```



Grafik API – OpenGL

- GL_TRIANGLES
  - Min. 3 Punkt
  - Vielfaches von 3

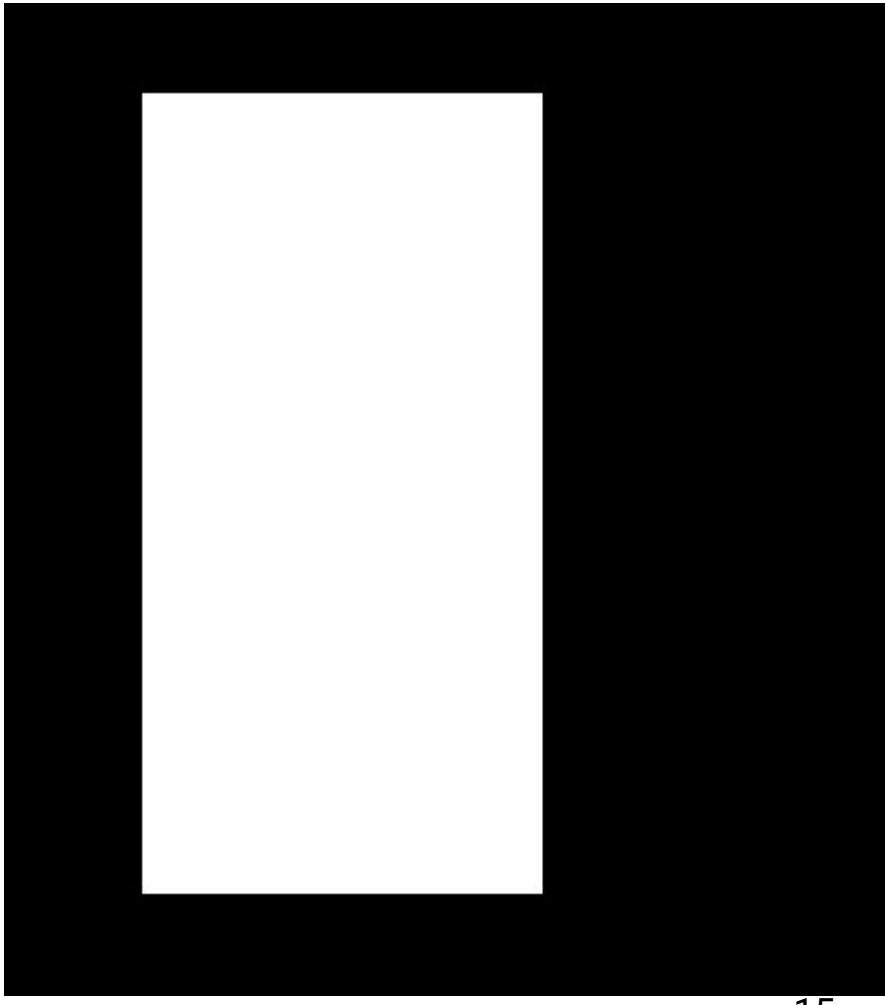
```
glBegin(GL_TRIANGLES);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);

glVertex2f(0, -0.5f);
glVertex2f(0.5f, 0.5f);
glVertex2f(0.5f, 0);

glVertex2f(0, 0);
glVertex2f(1, 1);
glEnd();
```



Grafik API – OpenGL

- GL_QUADS
  - Min. 4 Punkt
  - Vielfaches von 4

```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, -0.5f);
glVertex2f(0, 0.5f);
glEnd();
```



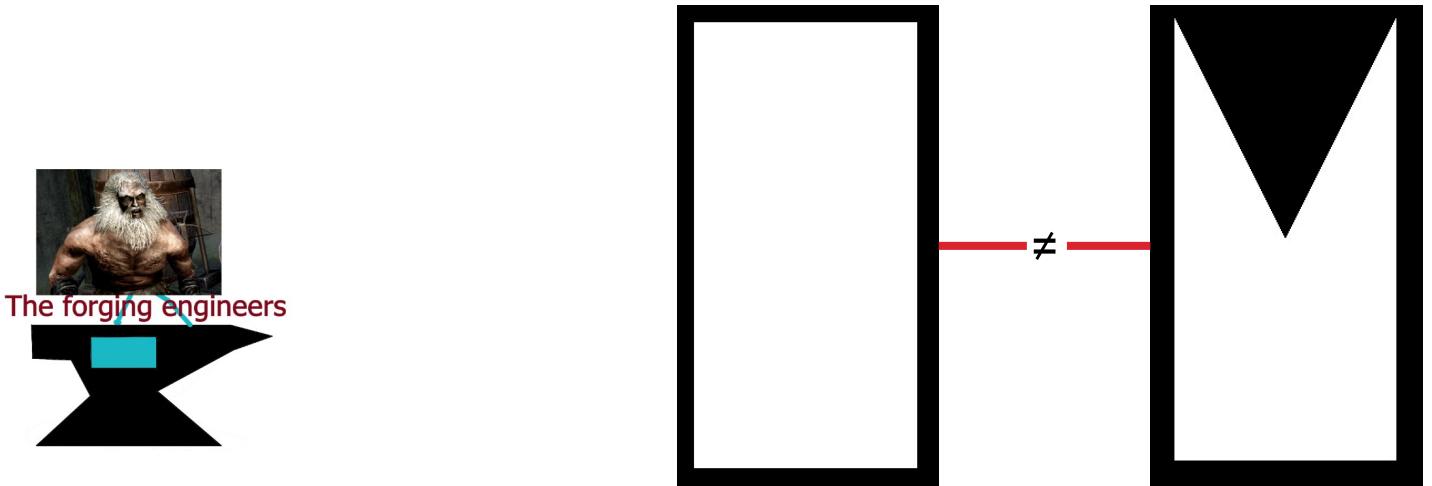
Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, -0.5f);
glVertex2f(0, 0.5f);
glEnd();
```

?
=

```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```

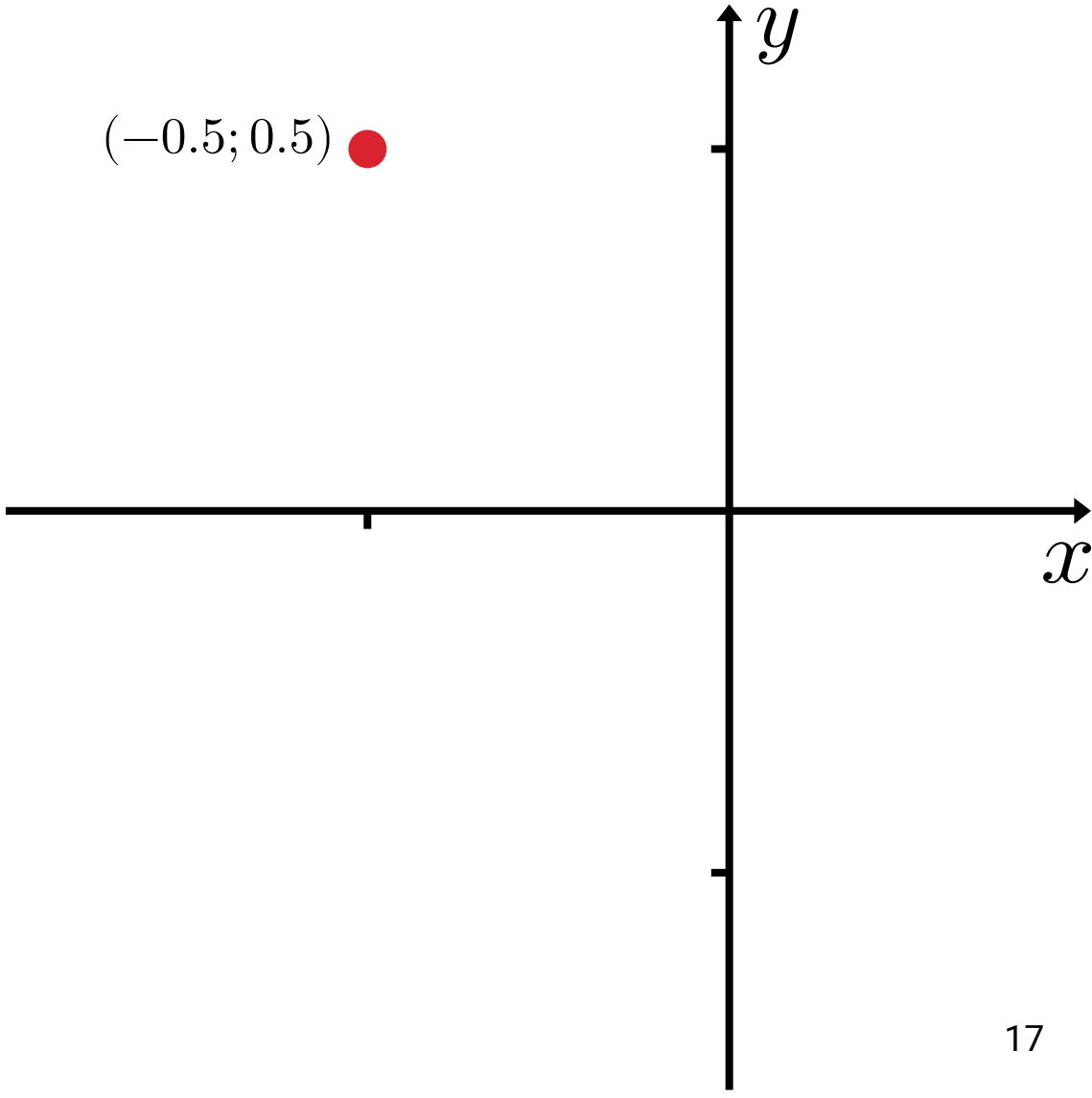


≠

Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

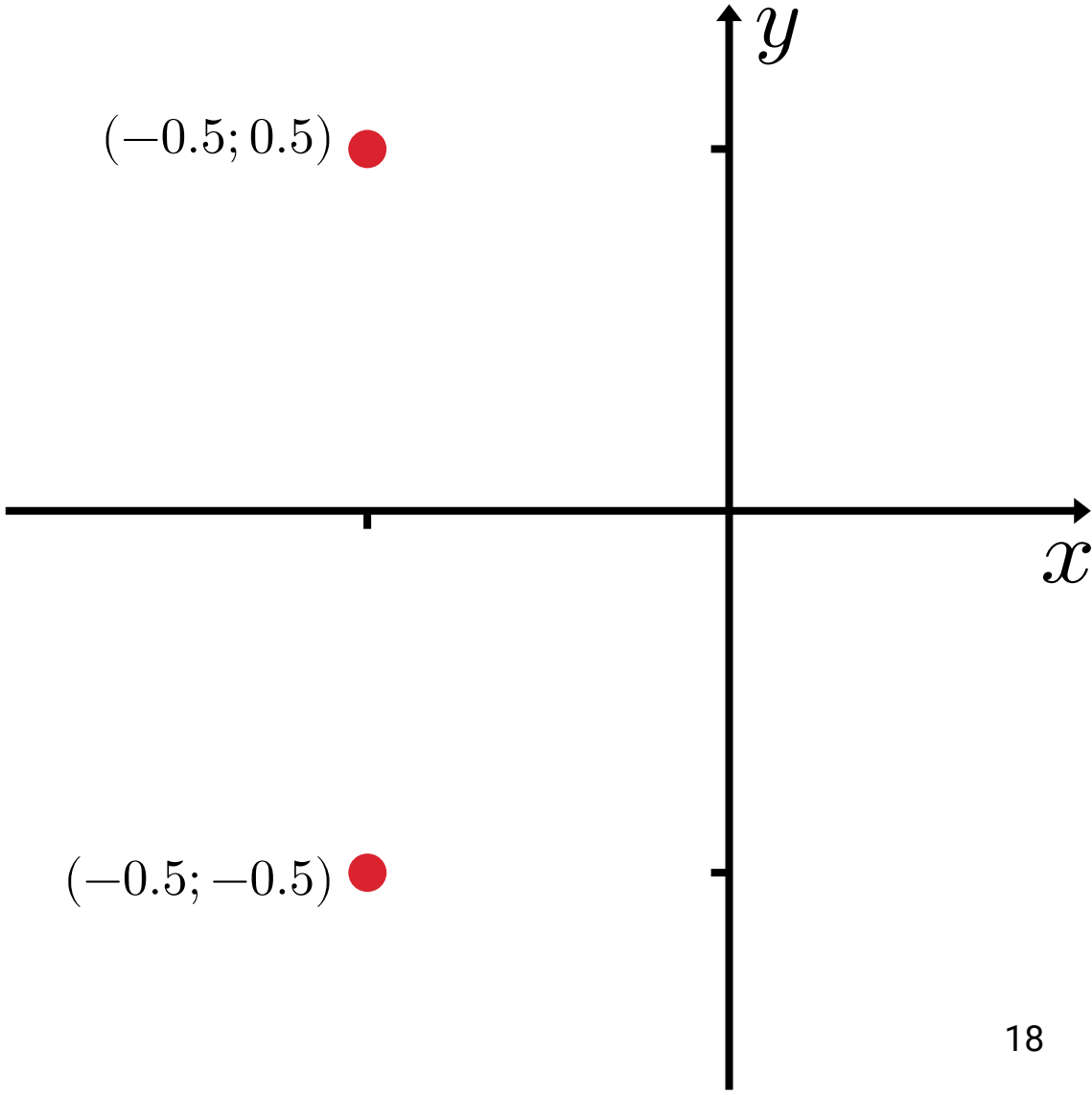
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

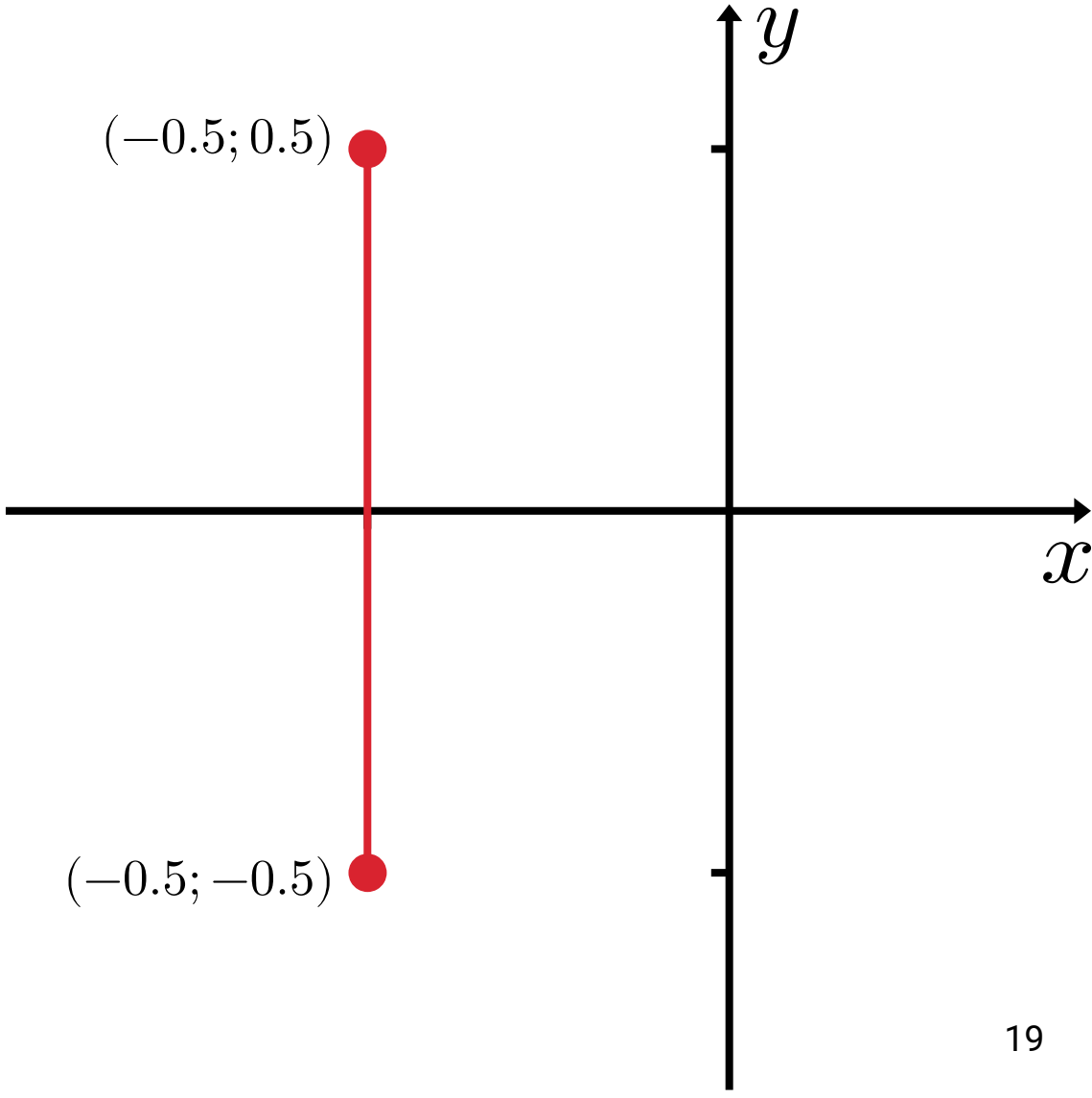
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

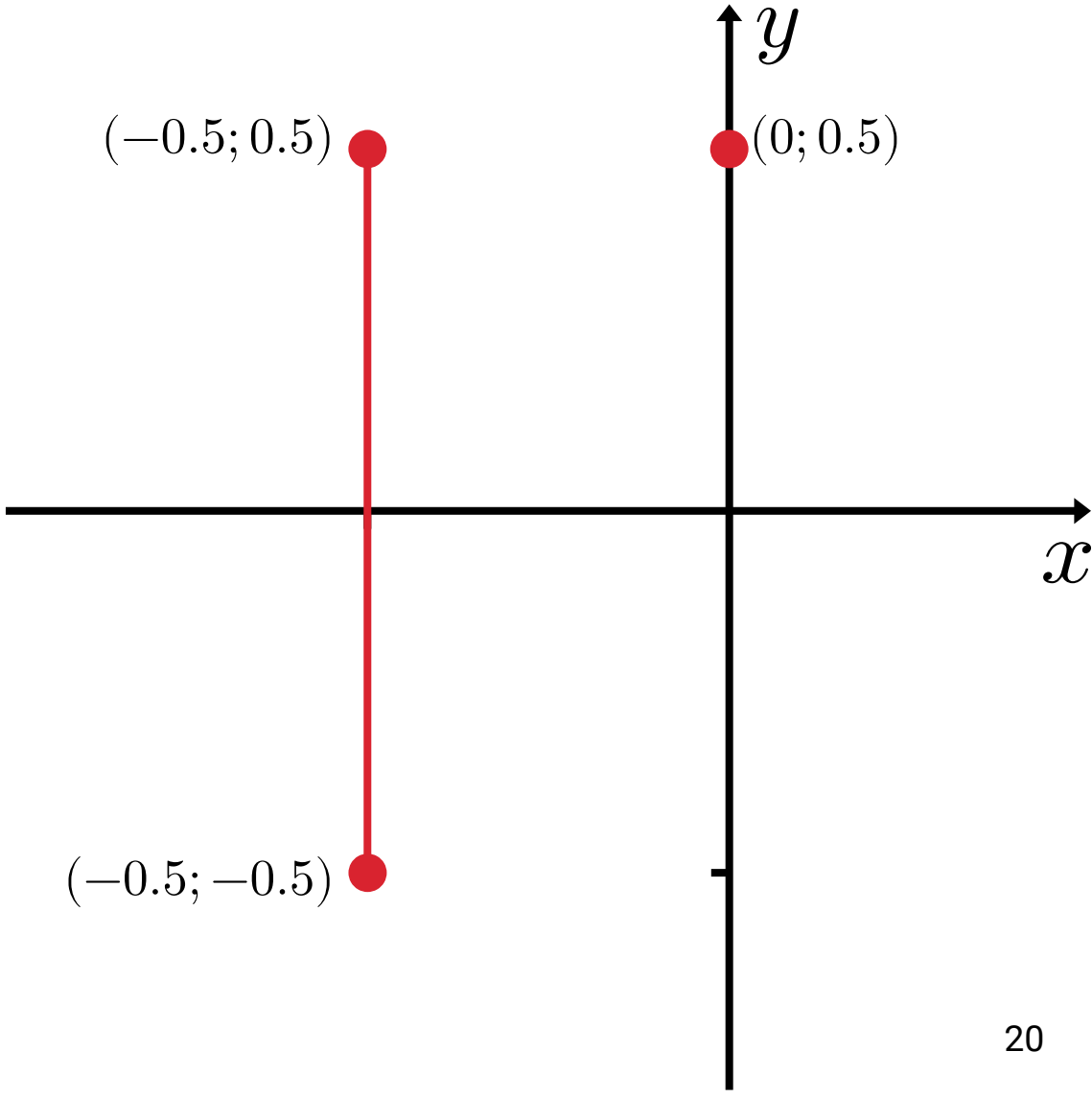
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

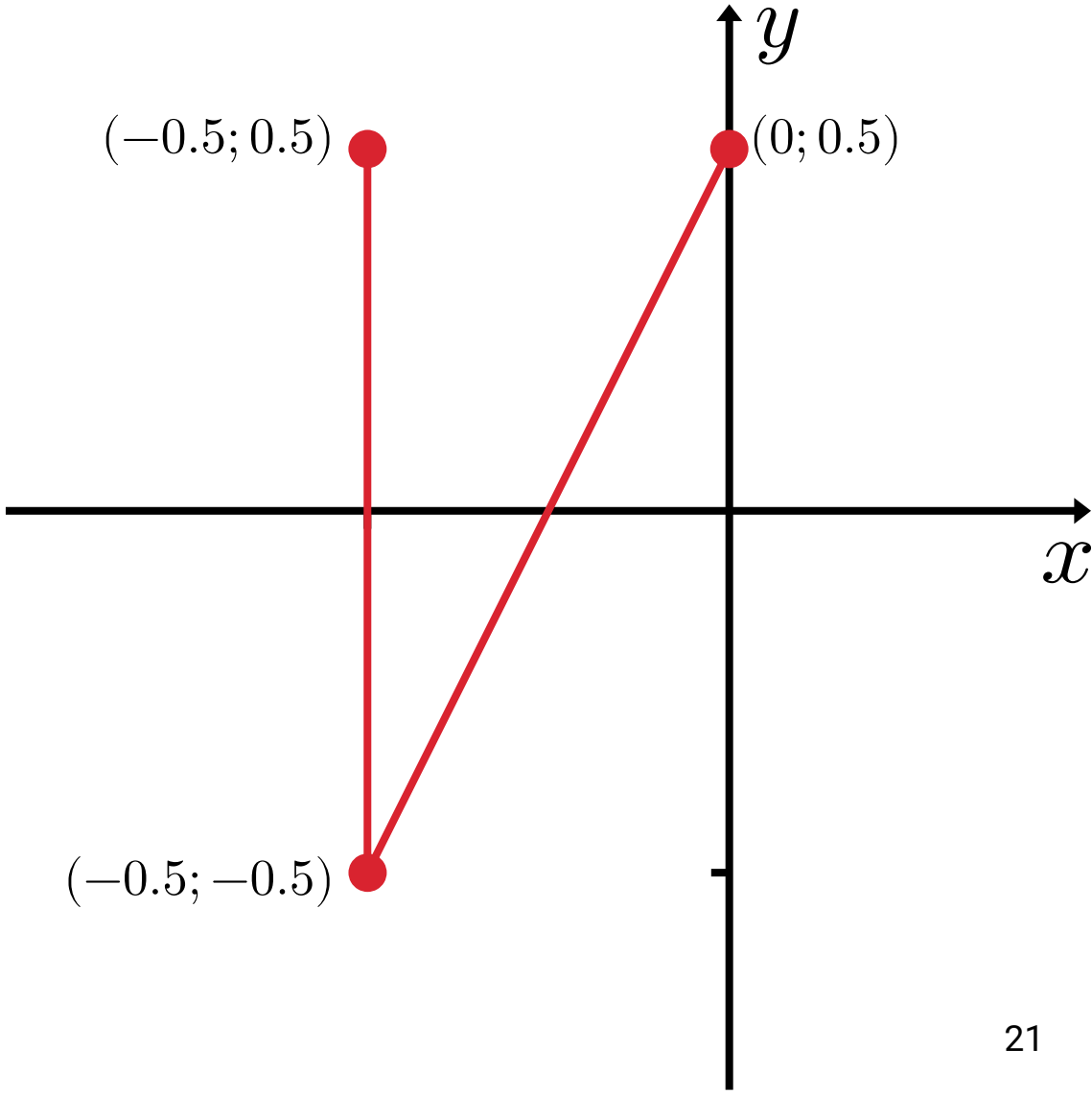
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

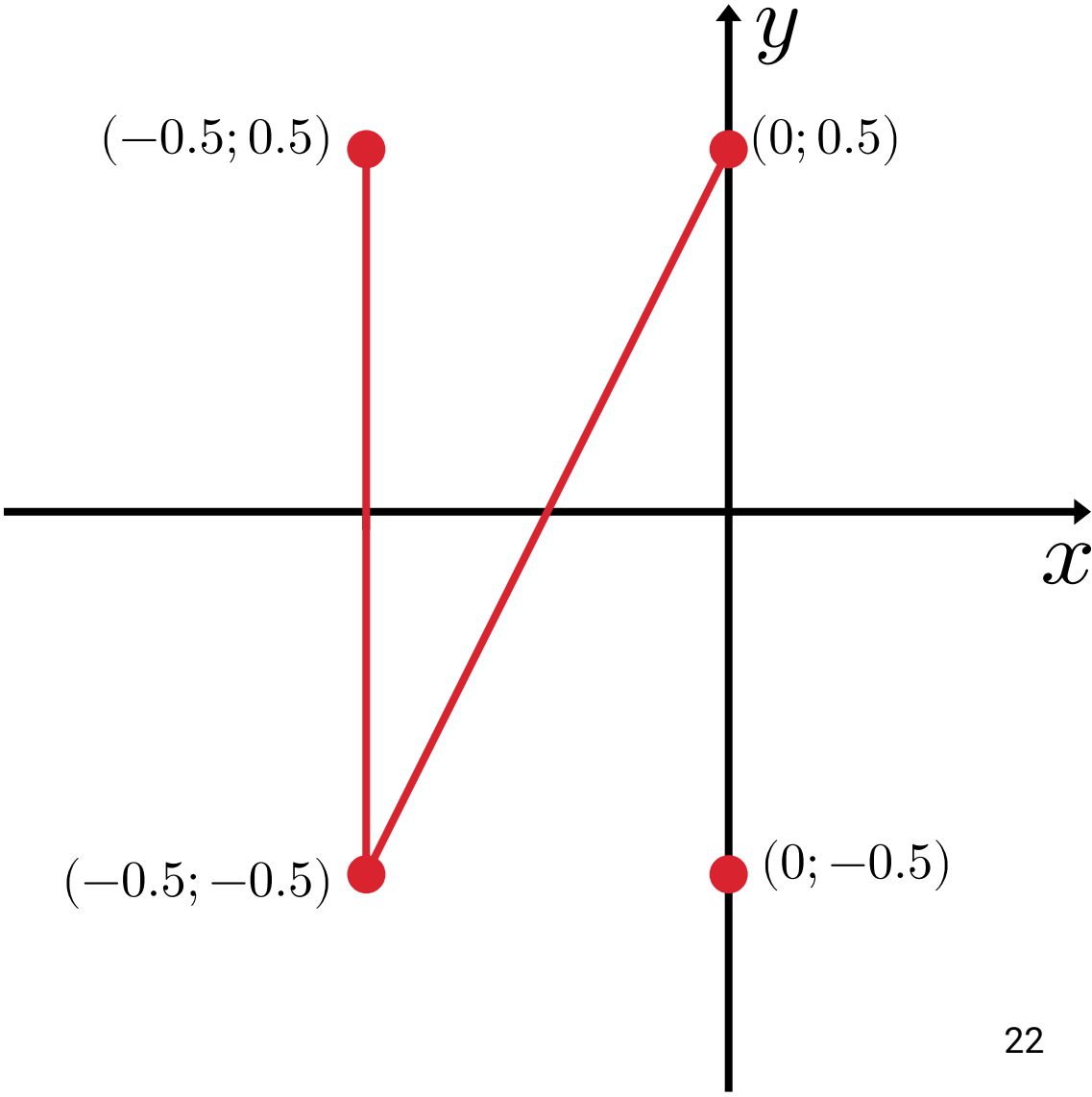
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

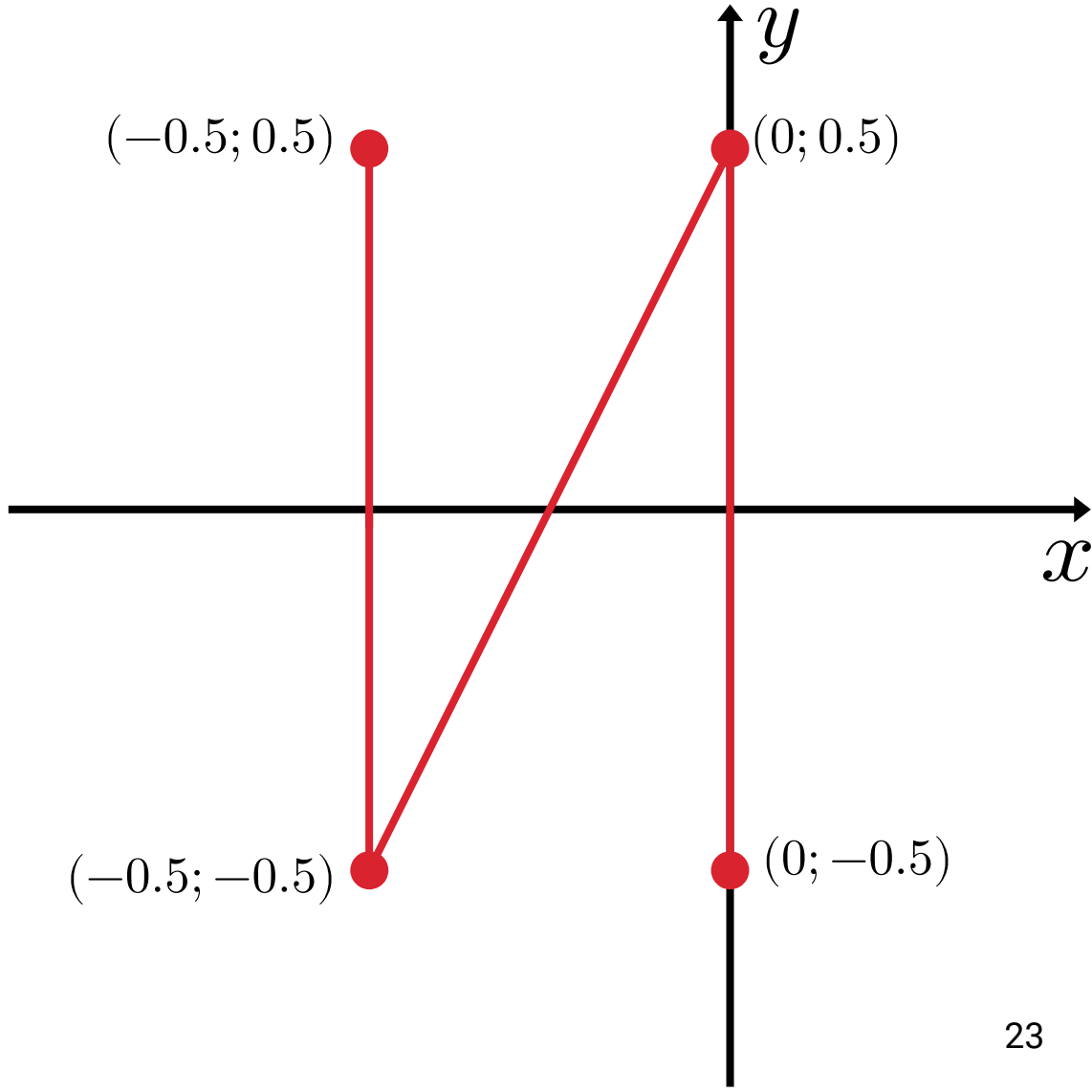
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

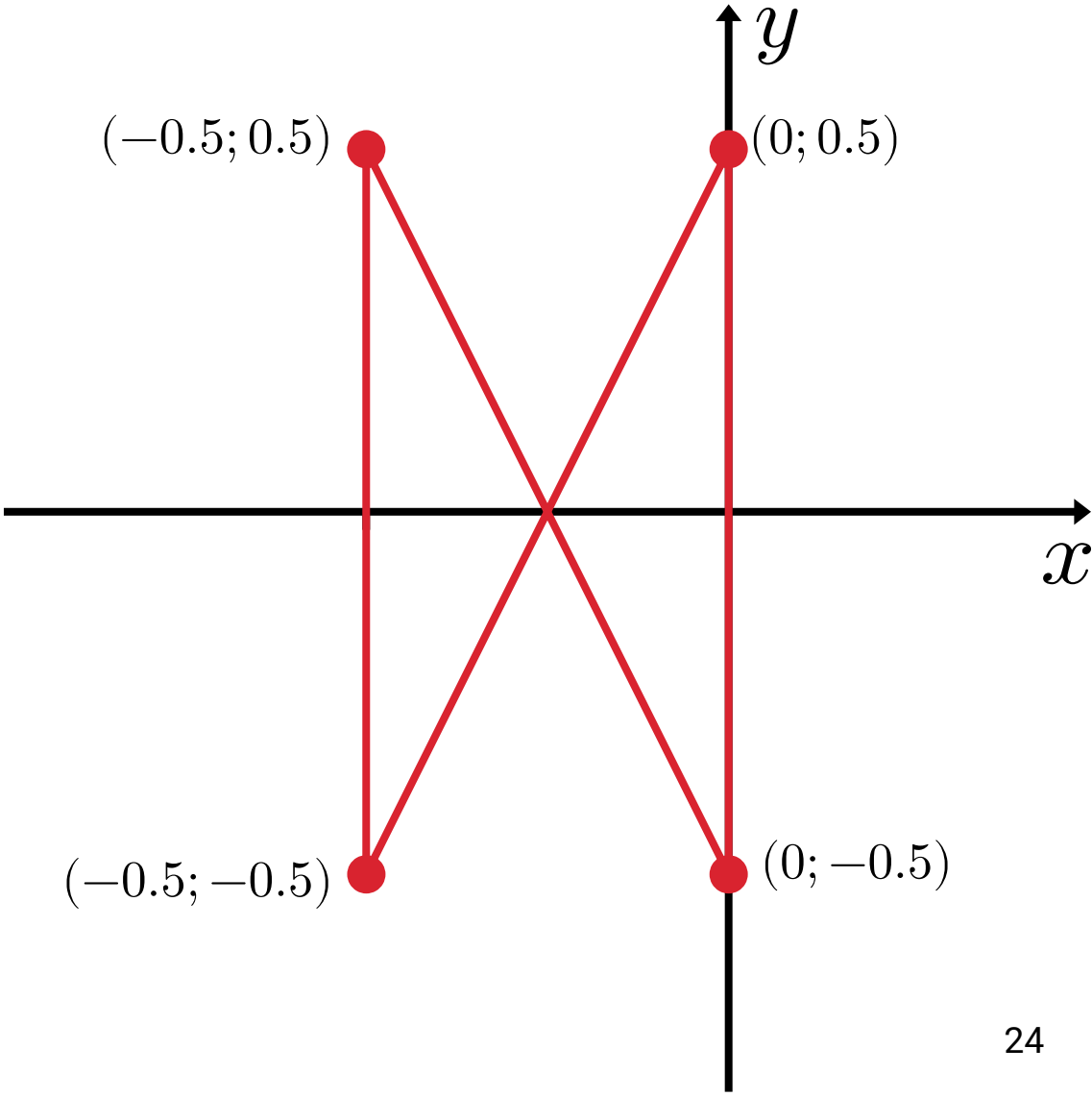
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

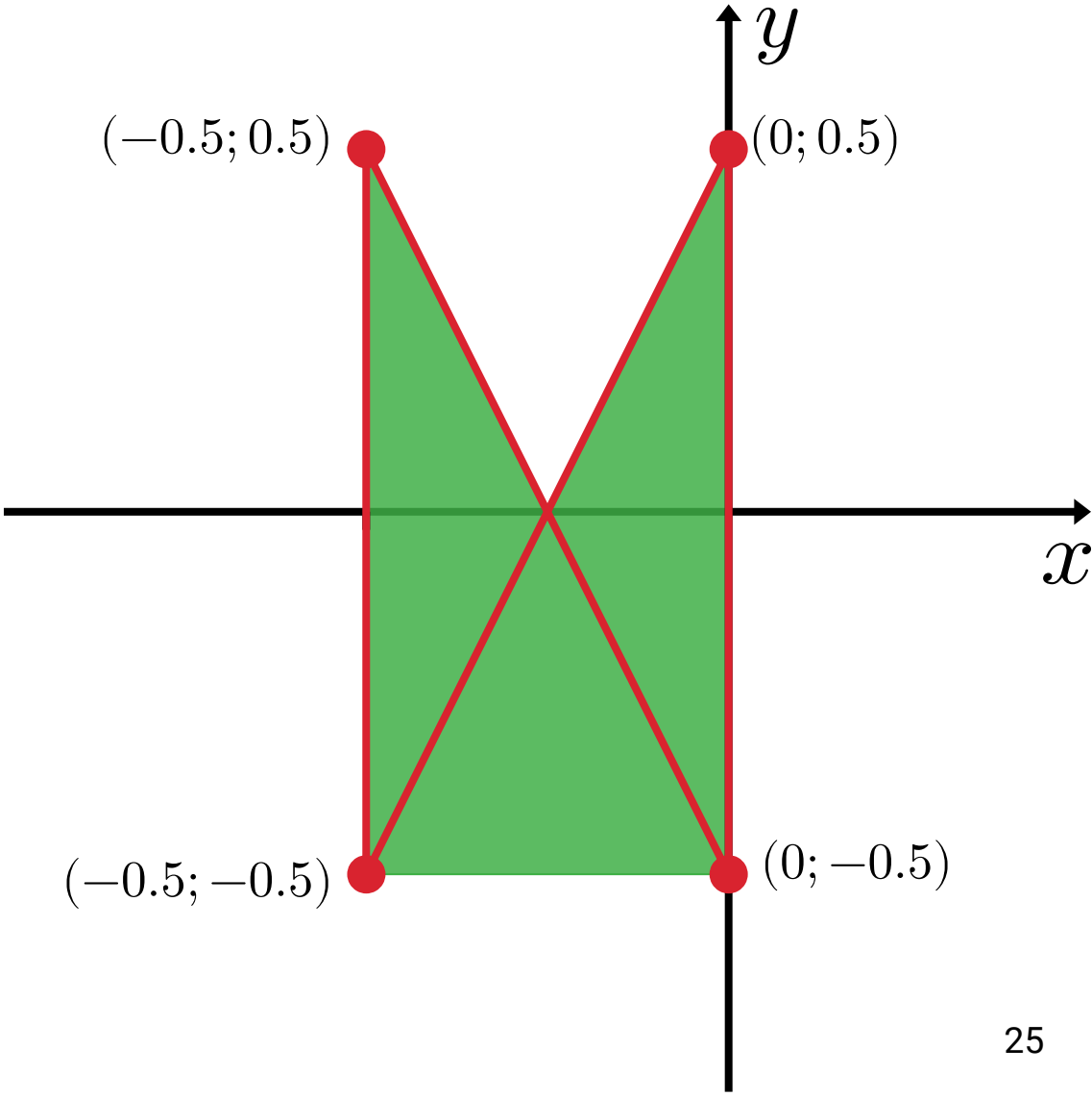
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

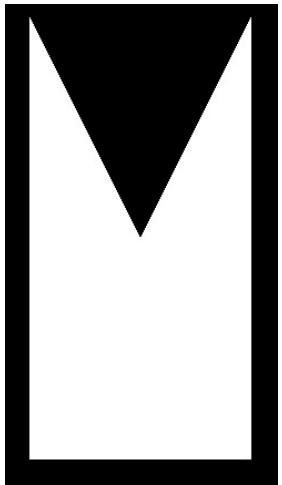
```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```

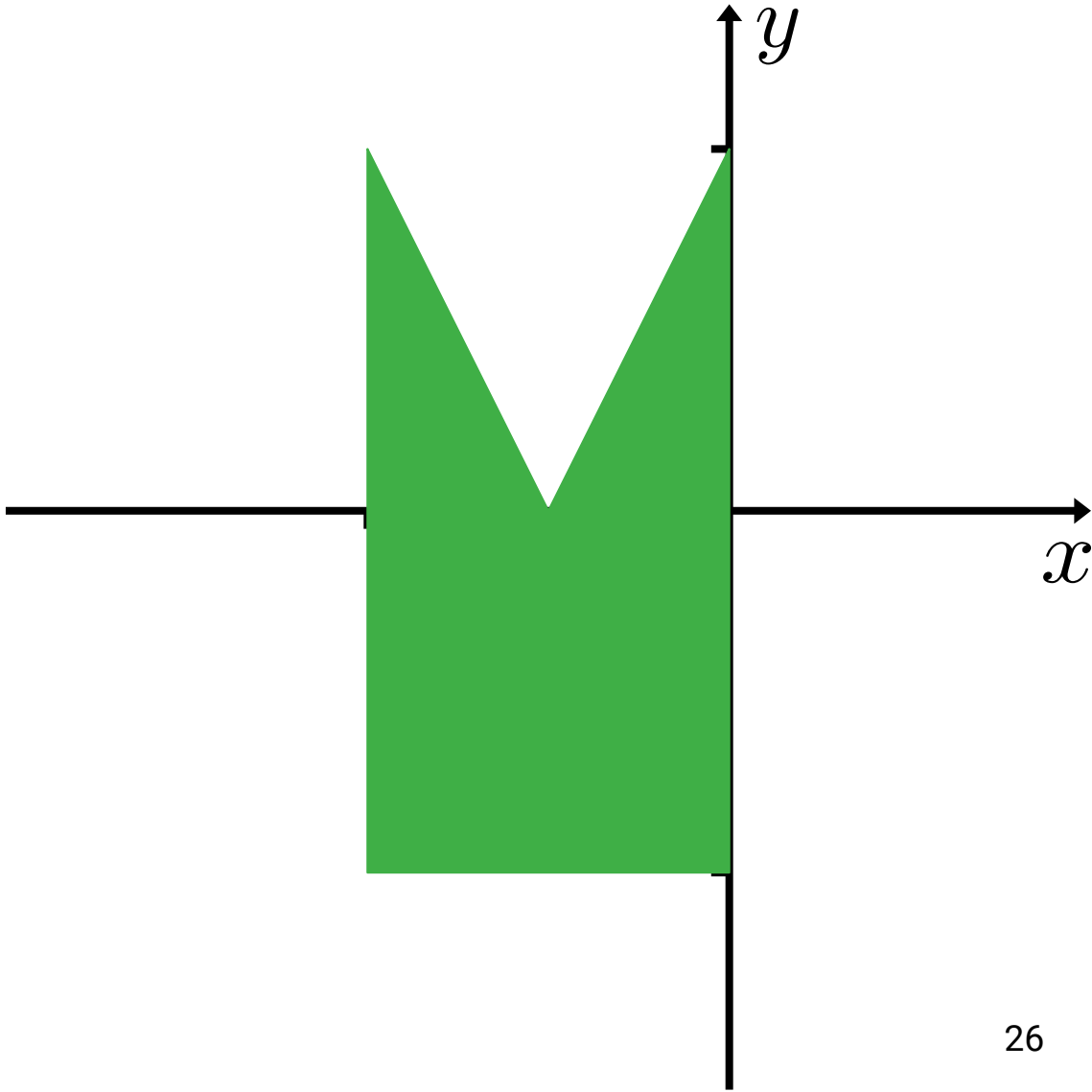


Grafik API – OpenGL

- Reihenfolge der Punkte wichtig
  - z.B. GL_QUADS

```
glBegin(GL_QUADS);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```



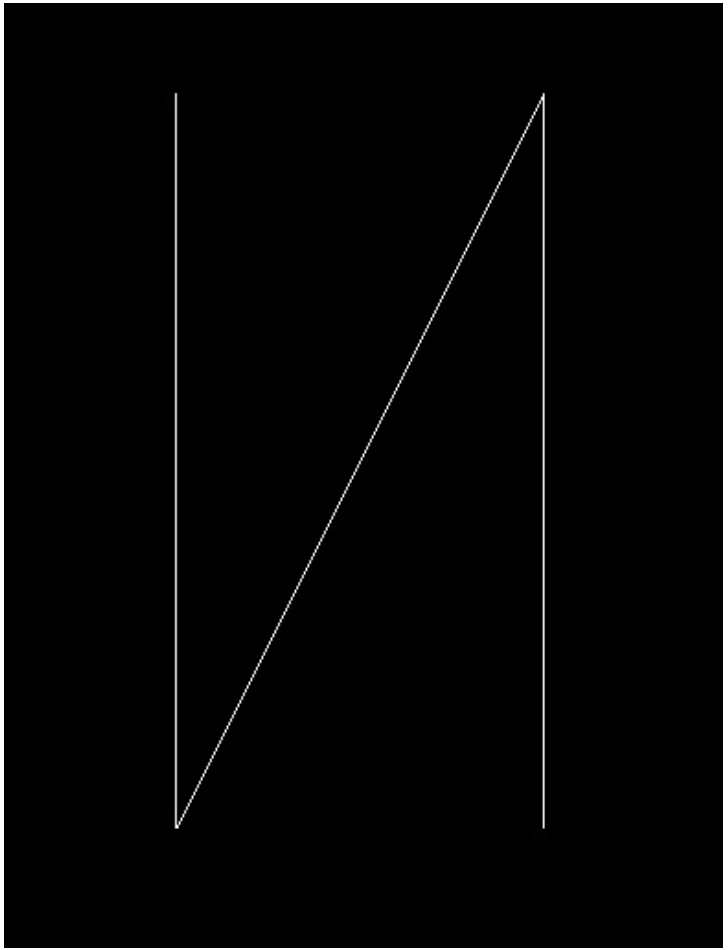


Grafik API – OpenGL

- GL_LINE_STRIP
  - Min. 2 Punkt
  - Kein Vielfaches

```
glBegin(GL_LINE_STRIP);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);
glVertex2f(0, -0.5f);
glEnd();
```

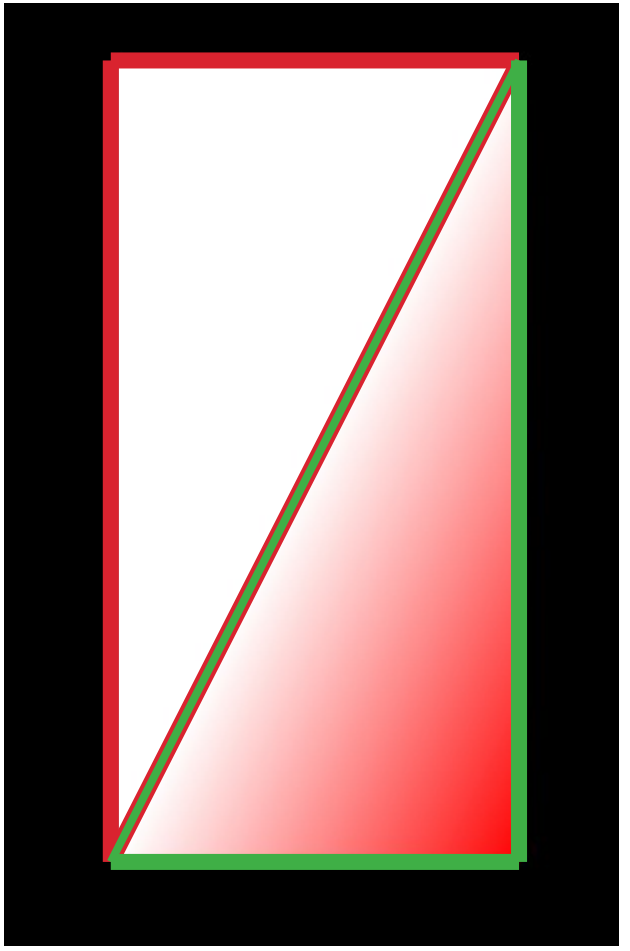
  - Jeder nachfolgende Punkt wird mit dem davor verbunden



Grafik API – OpenGL

- GL_TRIANGLE_STRIP
  - Min. 3 Punkt
  - Kein Vielfaches

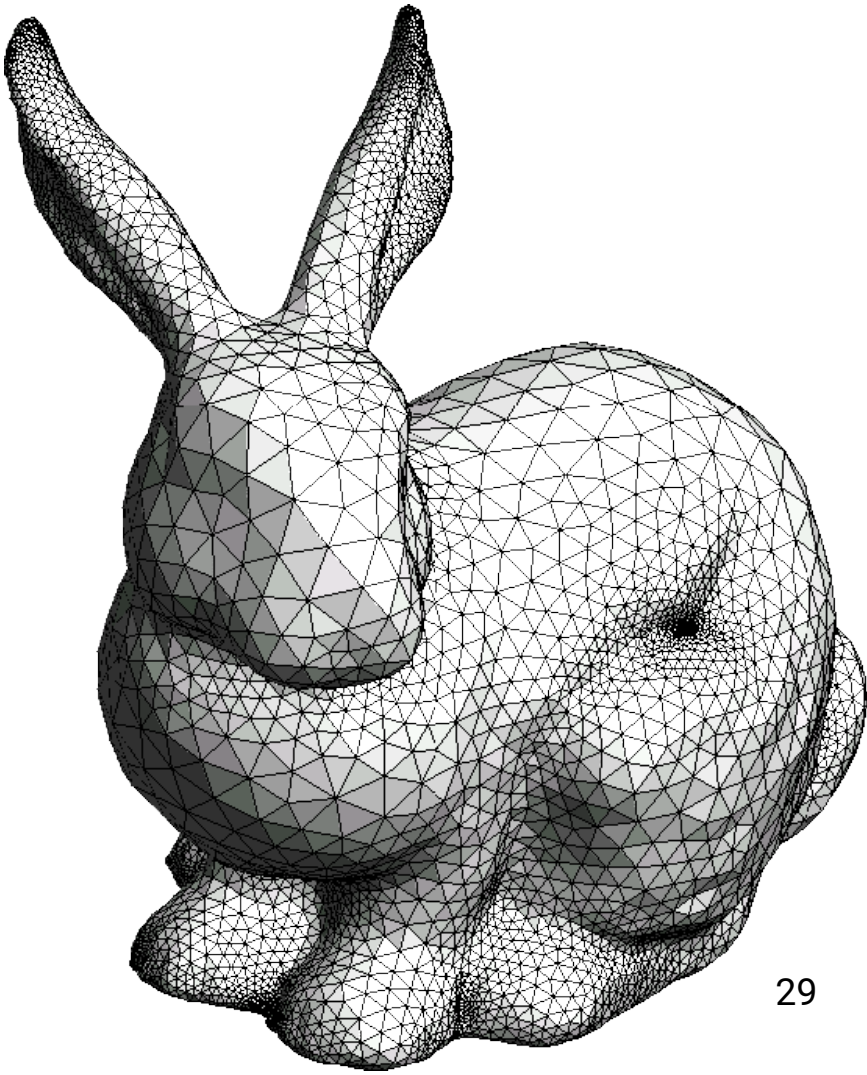
```
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(-0.5f, 0.5f);
glVertex2f(-0.5f, -0.5f);
glVertex2f(0, 0.5f);

glColor3f(1, 0, 0);
glVertex2f(0, -0.5f);
glEnd();
```



- Nächster Punkt bildet neuen Eckpunkt
  → eine Kante gemeinsam
  → zwei Eckpunkte gemeinsam
- Schnellster Modus! → Dreiecke sind Spezialgebiet von GPUs

Grafik API – OpenGL

- Meshes
  - Ein Mesh ist ein Gitter von Primitiven, die ein (3D-)Objekt modellieren
  - Häufig als Dreiecksgitter, da GPUs Dreiecke lieben
  - So ein Mesh könnte man bspw. mit *GL_TRIANGLE_STRIP* gut rendern



Ende

Fragen?