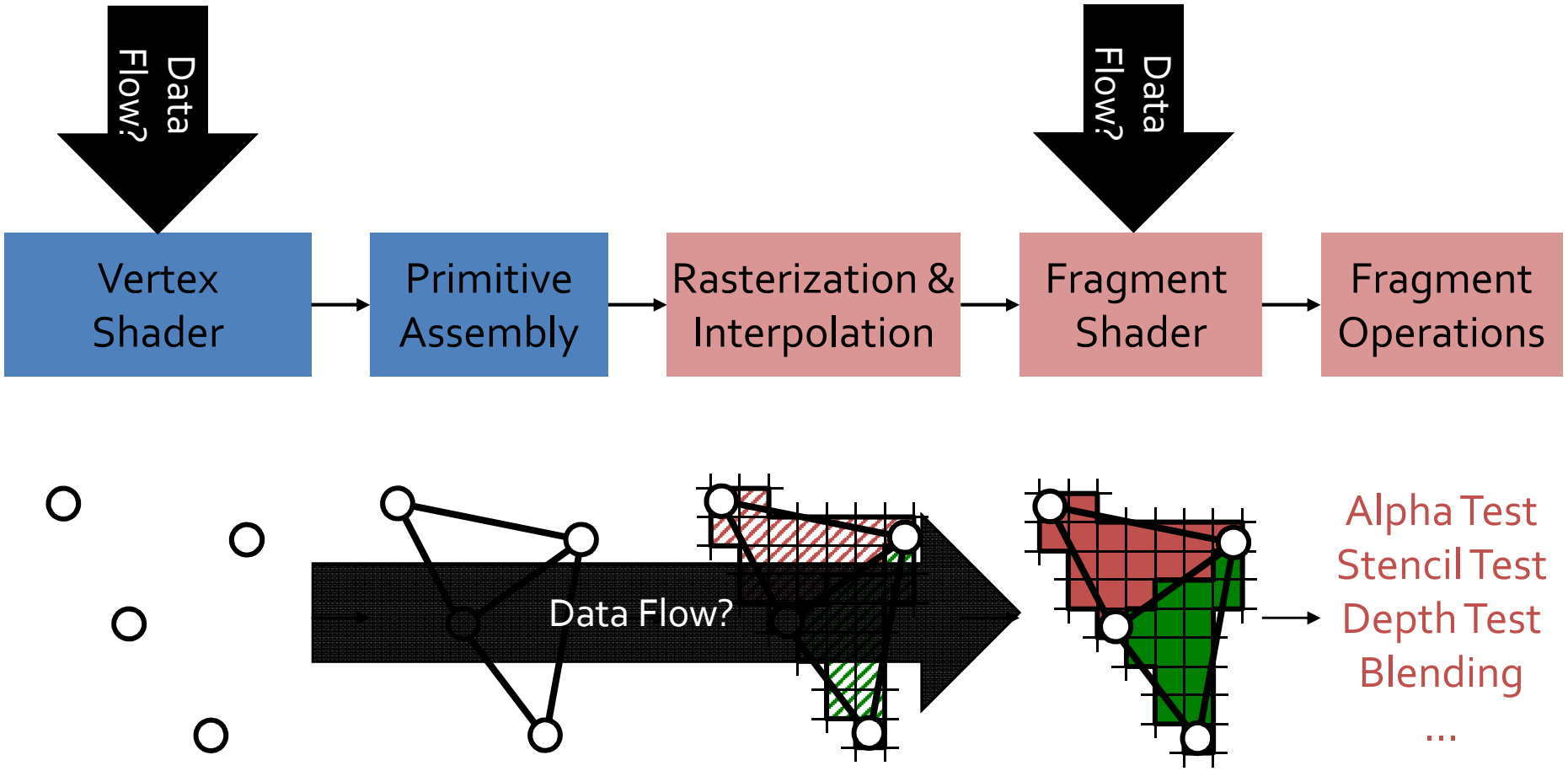
Shader

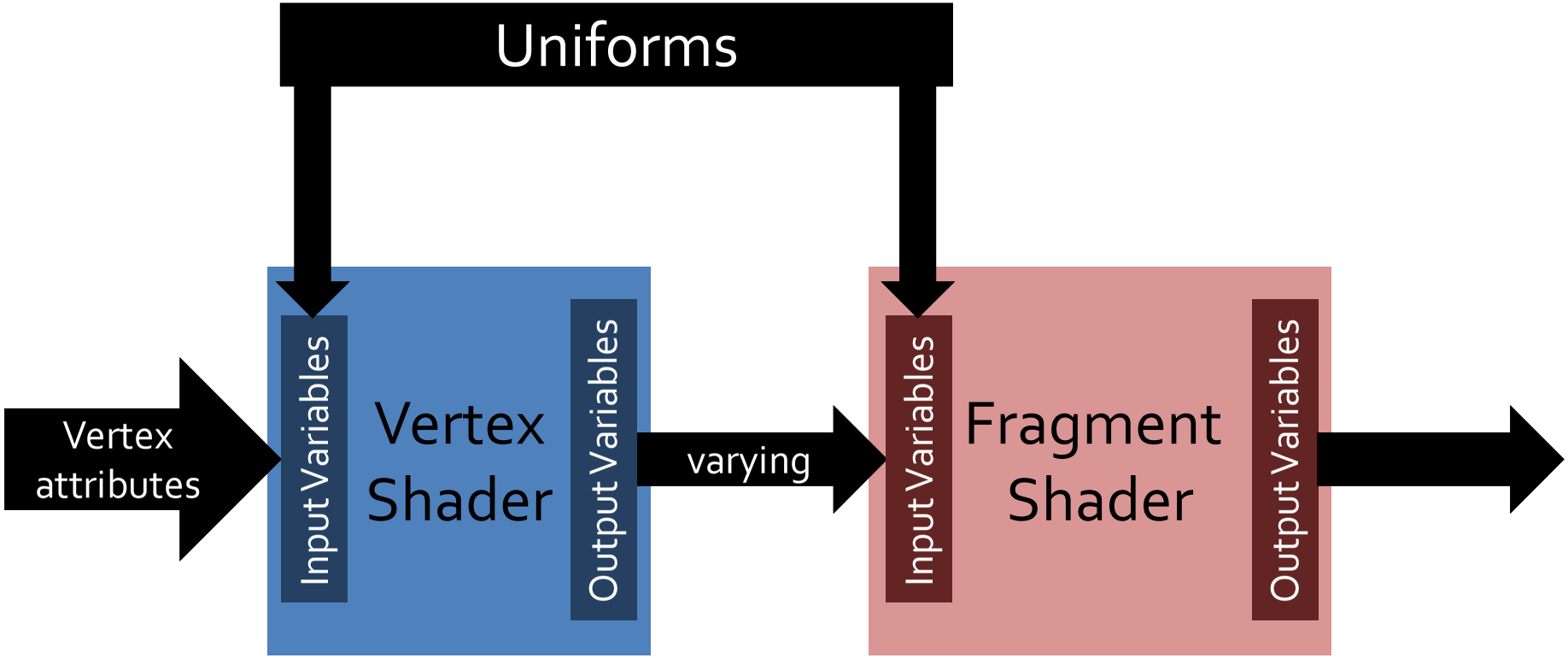
Pipeline

Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Programable Pipeline



Shader Data Flow



Uniform Variables

- **uniform** <datatype> <dataName>
- Values that remain constant during frame
- Read only in vertex and fragment shaders

```
uniform float time; // global time seconds
uniform mat4 MVP; // modelViewProjection
…
void main() {
  vec4 pos = position + velocity * time;
  gl_Position = MVP * pos;
}
```

Uniform Variables – OpenGL

- String names
- Different data types

```
Useprogram(shaderID);

var locTime = GetUniformLocation(shaderID, “time”);
Uniform1(locTime, 0.2f);

var locMVp = GetUniformLocation(shaderID, “MVP”);
Uniform1(locMVP, modelViewprojection);
```

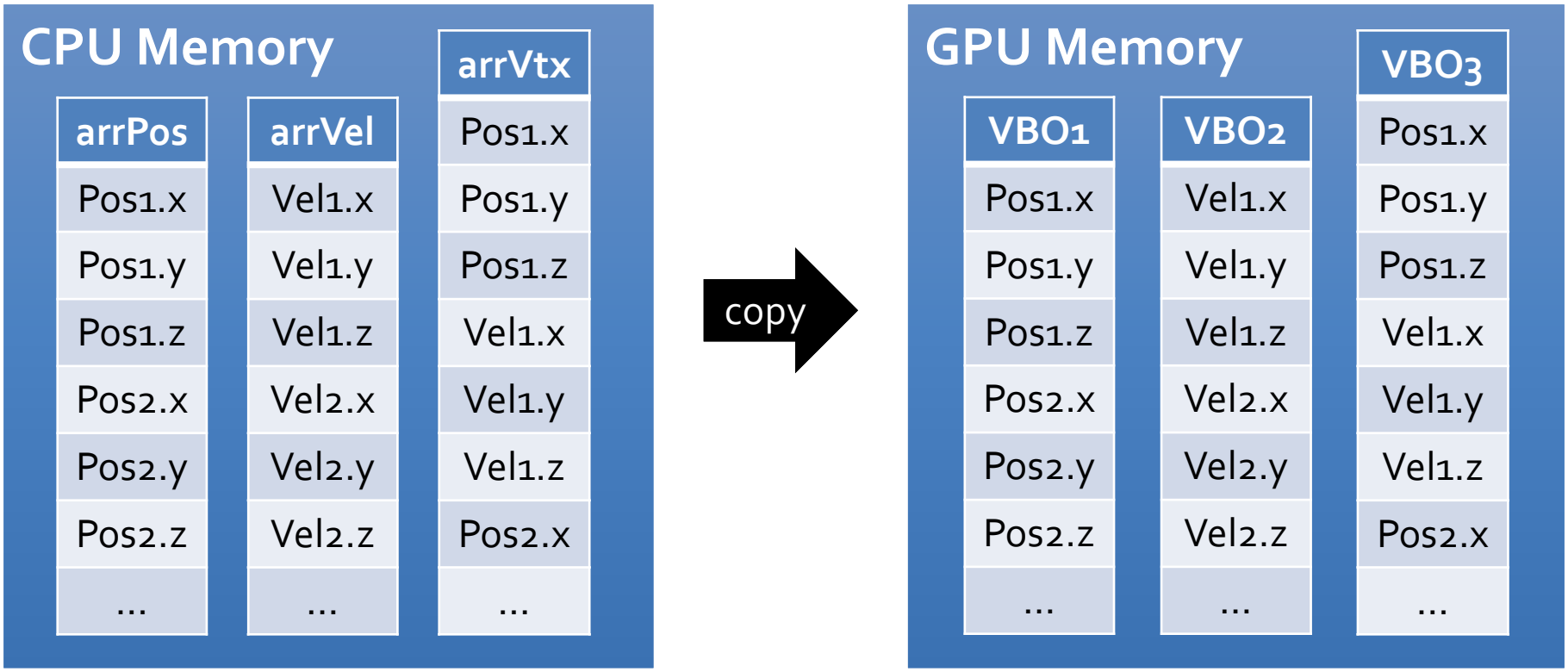
Vertex Attributes

- **`in <dataType>`** `<dataName>`
- Set variables **per vertex**
  - Read only in vertex shader
  - Fragment shader cannot access it

```
in vec4 position;
in vec4 velocity;
…
void main() {
  gl_Position = position + velocity * time;
}
```

Vertex Attributes – OpenGL (VBO)

- Set attributes of many vertices fast
- Chunk of memory containing vertex attributes for all vertices



Vertex Attributes – OpenGL (VBO)

- Vertex Buffer Objects = vertex data on hardware
- Create buffer object (get id)
- Set buffer data (copy from CPU)

```
// create buffer on hardware
uint bufferID;
GenBuffers(1, out bufferID);
BindBuffer(ArrayBuffer, bufferID);
// set buffer data
BufferData(ArrayBuffer, byteSizeData, ptrData);
```

Vertex Attributes – OpenGL (VBO)

- Activate before rendering
- Each vertex has associated active attribute array

```
EnableVertexAttribArray(id);
VertexAttribPointer(id, elementSize, elementDataType,
   isNormalized, stride, offset);
// render using active attributes
DrawArrays(PrimitiveType, firstVertex, vertexCount);
// deactivate attributes
DisableVertexAttribArray(id);
```

Vertex Attributes – OpenGL (VBO)

```
// attribute 0 is of type vec3
EnableVertexAttribArray(0);
VertexAttribPointer(0,3, float,false, sizeof(vec3),0);
// attribute 1 is of type vec2
EnableVertexAttribArray(1);
VertexAttribPointer(1,2, float,false, sizeof(vec2),0);
// render using active attributes
DrawArrays(PrimitiveType.Points, 0, vertexCount);
// deactivate attributes
DisableVertexAttribArray(1);
DisableVertexAttribArray(0);
```

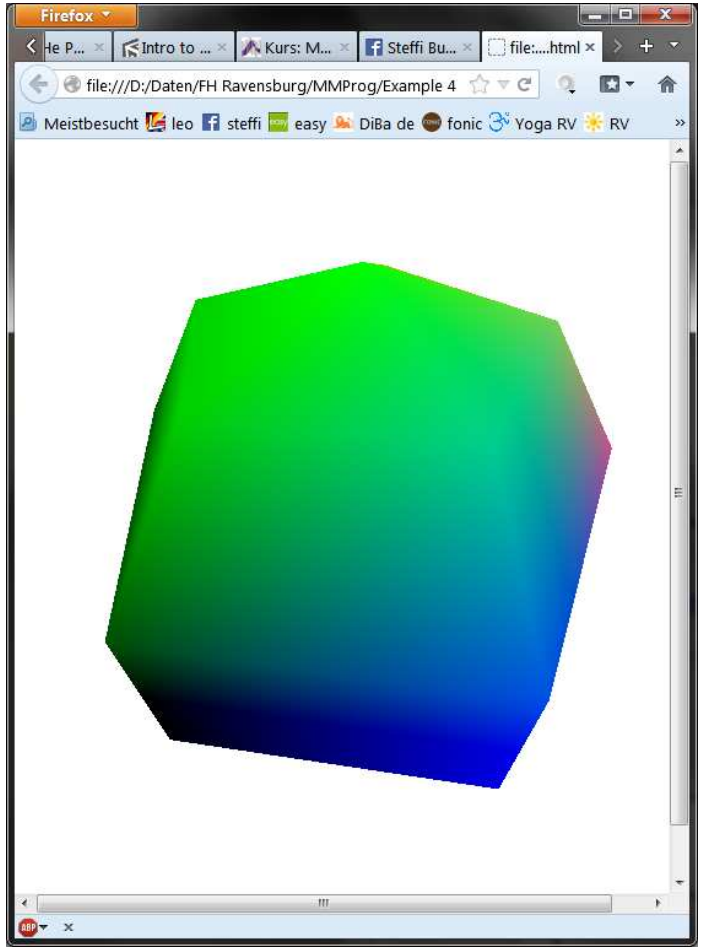
Varying Variables

- **`in/out <dataType>`** `<dataName>`
- Declare in vertex and fragment shader
  - Type and name must match
- Written in vertex shader
  - Read only in fragment shader
  - Fragment shader gets *interpolated* value

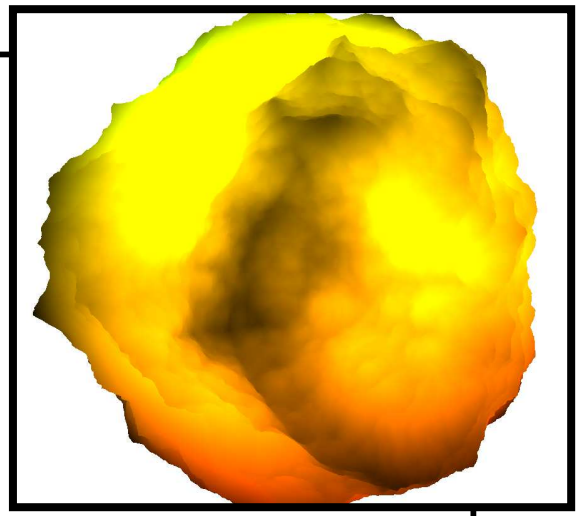
Varying Variables - Code

```
// vertex shader
varying vec3 pos;
void main() {
  pos = position;
  …
```

```
// fragment shader
varying vec3 pos;
void main() {
  gl_FragColor = vec4(pos, 1.0);
}
```



Displacement with Noise



```
// vertex shader
uniform float uTime;
varying float vNoise;
varying vec2 vUv;
void main() {
  vUv = uv;
  vNoise = -turbulence(0.5 *normal + uTime * 0.2);
  vec3 newPosition = position + normal * vNoise;
  …
```

```
// fragment shader
…
void main() {
  vec3 color = vec3(3.0*vUv*(1.0-2.0*vNoise),0.0);
  …
```

Samplers - Texture access code

```
// vertex shader
varying vec2 vUV;
void main() {
  vUV = uv; // model texture coordinates
  …
```

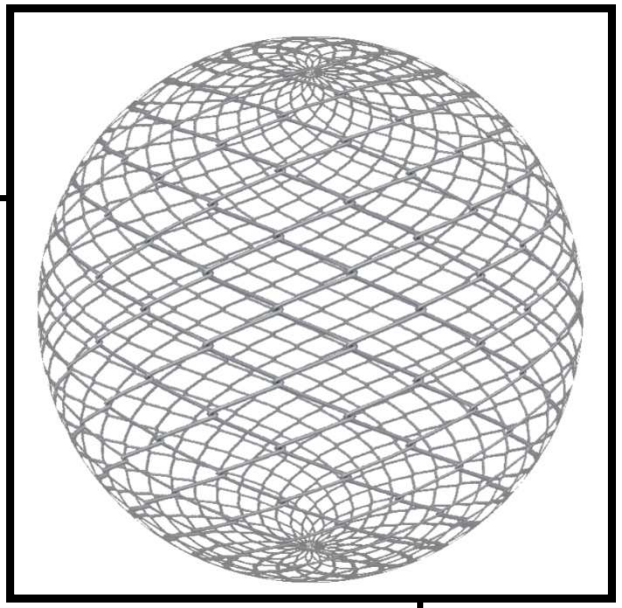
```
// fragment shader
uniform sampler2D tex;
varying vec2 vUV;
void main() {
  vec3 color = texture2D(tex, vUV).rgb;
  gl_FragColor = vec4(color.rgb, 1.0);
}
```

Samplers - Texture access code

```
// vertex shader
varying vec2 vUV;
void main() {
  vUV = uv; // model texture coordinates
  …
```

```
// fragment shader
uniform sampler2D tex;
varying vec2 vUV;
void main() {
  vec3 color = texture2D(tex, vUV).rgb;
  gl_FragColor = vec4(color.rgb, 1.0);
}
```

Alpha Texture



```
// fragment shader
uniform sampler2D tex;
varying vec2 vUV;
…
void main() {
   …
   vec4 color = texture2D(tex, vUV).rgba;
   if(0.05 > color.a) discard;
   …
}
```