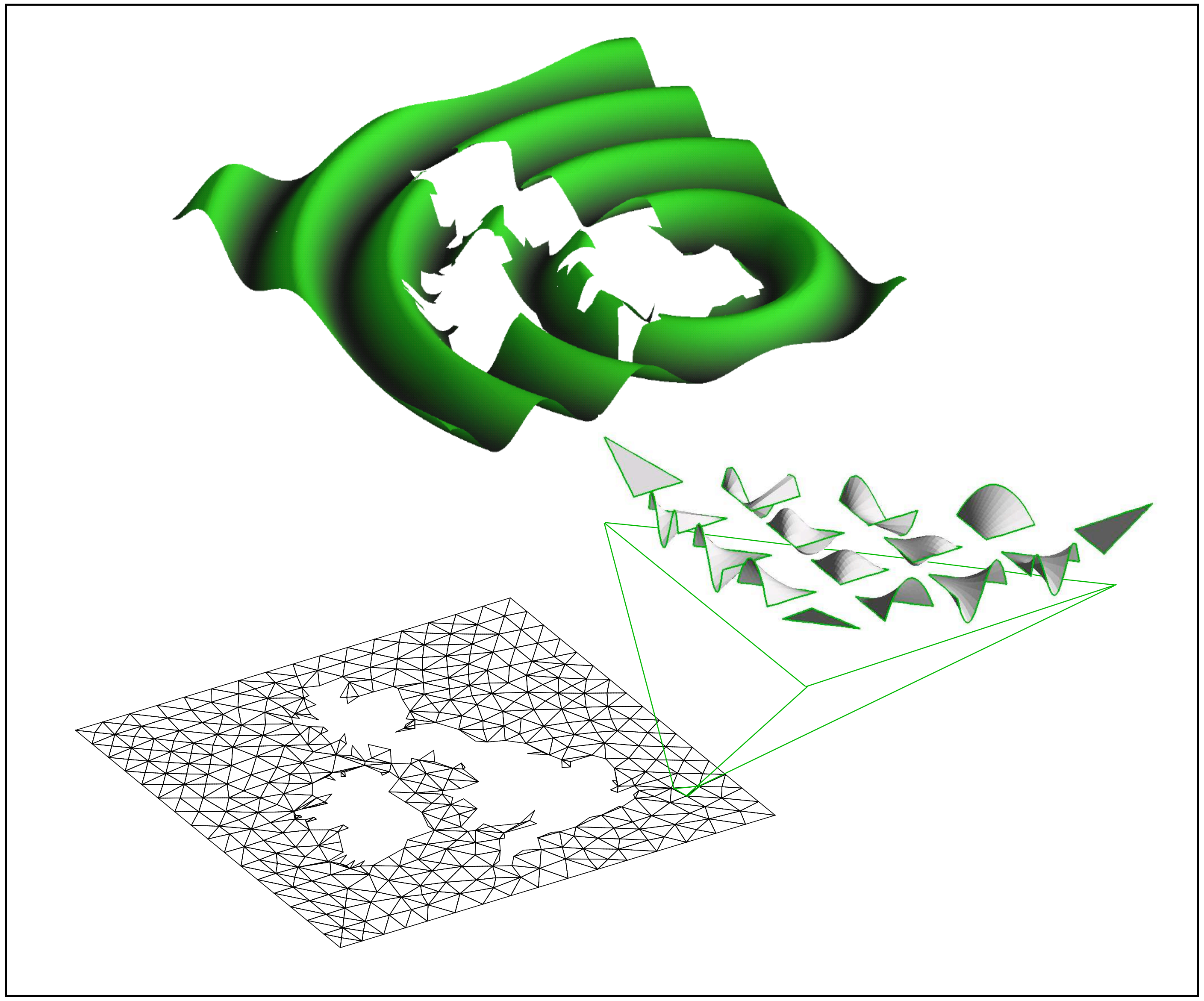
Computing Labs



Nektar++

Outline

- assignments
- find
- write
- build
- run
- verify

assignments

- handouts
  - tutorial 1:
    integration, differentiation & interpolation on a one-dimensional standard region
  - tutorial 2:
    towards a two-dimensional projection problem
  - tutorial 3:
    a 2D Helmholtz solver in Nektar++

➡ complete source code

find

- Visual studio 2005

write

- visual studio build in editor
- C++ programming language
- Nektar++ syntax

write: C++

- Fundamental data types

```
int foo;
int foo = 2;

double foo;
double foo = 2.1;

NekDouble foo;
NekDouble foo = 2.1;
```

- Comments

```
\\ This is a comments inside the code
```

- Loops

```
int i;

for(i = min; i < max; i++)
{
    \\ your implementation
}
```

write: C++

- Fundamental data types

```
int foo;
int foo = 2;

double foo;
double foo = 2.1;

NekDouble foo;
NekDouble foo = 2.1;
```

- Comments

```
\\ This is a comments inside the code
```

- Loops

```
int i;                                i=i+1;

for(i = min; i < max; i++)
{
    \\ your implementation
}
```

write: C++

- Mathematical expressions <cmath>

```
pow(x,7)
sin(x)
cos(x)
```

write: Nektar++

- Array
  - Array<OneD, NekDouble>

    ```
    Array<OneD, NekDouble> foo(size);
    Array<OneD, NekDouble> foo(size,value);
    ```

  - element access

    ```
    foo[i]
    ```

  - index starts with zero

    ```
    for(i = 0; i < size; i++)
    {
        foo[i] = ...
    }
    ```

  - efficient allocation
  - automatic dealocation

write: Nektar++

- data managers
  - PointsManager
  - BasisManager
  - data key

```
int Q = 4;
LibUtilities::PointsType type = LibUtilities::eGaussLobattoLegendre;
const LibUtilities::PointsKey key(Q, type)

Array<OneD, NekDouble> quadZeros(size);
quadZeros = (LibUtilities::PointsManager()[key])->GetZ();
```

- NekMatrix
- NekVector

build - run - verify

Nektar++