Beyond PLY

Daniel Pfeifer

VMML@IFI Group Meeting on July 1, 2011

▷ Random Access PLY
PLY 1.0
PLY 1.0
PLY 2.0
PLY 2.0
Why PLY?

Random Access PLY

PLY 1.0

Random Access PLY
▷ PLY 1.0
PLY 1.0
PLY 2.0
PLY 2.0
Why PLY?

```
ply
format ascii 1.0
comment made by anonymous
comment this file is a cube
element vertex 8
property float32 x
property float32 y
property float32 z
element face 6
property list uint8 int32 vertex_index
end_header
0 0 0
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

PLY 1.0

Random Access PLY
PLY 1.0
▷ PLY 1.0
PLY 2.0
PLY 2.0
Why PLY?

- □ format := ascii | binary
- □ element := name size property+
- □ property := name (scalar | list)
- □ list := scalar scalar

Problem:
Variable sized content limits parsing to **forward** only!
There is no way to implement **random access**.
(Analogous to **Forward** resp. **RandomAccess** iterator concepts).

PLY 2.0

Random Access PLY
PLY 1.0
PLY 1.0
▷ PLY 2.0
PLY 2.0
Why PLY?

- format is always binary, native byteorder
- dropped support for lists
- new support for **arrays**
- array := size scalar

List:
property array vertex_indices uint8 uint32

Array:
property array vertex_indices 3 uint32

PLY 2.0

Random Access PLY
PLY 1.0
PLY 1.0
PLY 2.0
▷ PLY 2.0
Why PLY?

```
ply
format ascii 2.0
comment made by anonymous
comment this file is a cube
element vertex 8
property float32 x
property float32 y
property float32 z
element face 6
property array 4 int32 vertex_index
end_header
0 0 0
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
0 1 2 3
7 6 5 4
0 4 5 1
1 5 6 2
2 6 7 3
3 7 4 0
```

Random Access PLY

▷ Why PLY?

Why PLY?
easy and simple?
easy and simple?
simplify
Qi
easy and simple?
new grammar
easy and simple?
Bonus

Why PLY?

Why PLY?

Random Access PLY

Why PLY?

▷ Why PLY?
easy and simple?
easy and simple?
simplify
Qi
easy and simple?
new grammar
easy and simple?
Bonus

Quoting the original PLY doc:

“Our goal is to provide a format that is **simple and easy** to implement but that is general enough to be useful for a wide range of models.”

- □ Was that goal achieved?
- □ Is it easy to implement?
- □ Is it simple to implement?

easy and simple?

Random Access PLY

Why PLY?
Why PLY?
▷ easy and simple?
easy and simple?
simplify
Qi
easy and simple?
new grammar
easy and simple?
Bonus

All the identifiers are at the beginning of a line.
It is possible to parse it like this:

- □ while(not EOF)
  - – read line
  - – tokenize line
  - – if token == ”format”
    - ▷ ...
  - – else if token == ”comment”
    - ▷ ...
  - – ...

easy and simple?

Random Access PLY
Why PLY?
Why PLY?
easy and simple?
▷ easy and simple?
simplify
Qi
easy and simple?
new grammar
easy and simple?
Bonus

- □ only minimal programming skills required
  $\rightarrow$ easy: YES
- □ Lots of code required:
  $\rightarrow$ simple: NO!!

simplify

Random Access PLY

Why PLY?
Why PLY?
easy and simple?
easy and simple?
▷ simplify
Qi
easy and simple?
new grammar
easy and simple?
Bonus

Generate parser from formal grammar:

```
ply       ::= "ply" EOL "format" format DOUBLE EOL element*
element   ::= "element" STRING INT EOL property*
property  ::= "property" (list | scalar) STRING EOL
list      ::= "list" size scalar
format    ::= "ascii"|"binary_little_endian"|"binary_big_end
size      ::= "uint8"|"uint16"|"uint32"|"uint64"
scalar    ::= size|"int8"|"int16"|"int32"|"int64"|"float32"|
```

Qi

Random Access PLY

Why PLY?
Why PLY?
easy and simple?
easy and simple?
simplify
▷ Qi
easy and simple?
new grammar
easy and simple?
Bonus

C++ implementation with Qi (Boost.Spirit):

```
start %= qi::eps > "ply" > qi::eol
  > "format" > format_ > qi::double_ > qi::eol
  > *element_
  > "end_header" > qi::eol;

element_ %= "element"
  > *(ascii::char_ - qi::int_)
  > qi::int_ > qi::eol
  > *property_;

property_ %= "property" > (list_ | scalar_)
  > *(ascii::char_ - qi::eol) > qi::eol;

list_ %= "list" > size_ > scalar_;
```

easy and simple?

Random Access PLY

Why PLY?

Why PLY?

easy and simple?

easy and simple?

simplify

Qi

▷ easy and simple?

new grammar

easy and simple?

Bonus

- □ knowledge about formal languages required (2nd semester)
  $\rightarrow$ easy: YES
- □ seven rules for the complete PLY grammar:
  $\rightarrow$ simple: YES

But this would hold true for any grammar!
So let’s change it...

new grammar

Random Access PLY

Why PLY?
Why PLY?
easy and simple?
easy and simple?
simplify
Qi
easy and simple?
▷ new grammar
easy and simple?
Bonus

```
start
  %= qi::eps
  > "#define" > endian
  > *element_
  > qi::eoi
  ;
element_
  %= "typedef struct {"
  > *attribute_
  > '}' > string_ > ',' > string_ > size_ > ';'
  ;
attribute_
  %= scalar_ > string_ > size_ > ';'
  ;
string_
  %= qi::lexeme[+(ascii::alnum | qi::char_('_'))]
  ;
size_
  %= ('[' > qi::uint_ > ']') | qi::eps(qi::_val = 1)
  ;
```

easy and simple?

Random Access PLY

Why PLY?
Why PLY?
easy and simple?
easy and simple?
simplify
Qi
easy and simple?
new grammar
▷ easy and simple?
Bonus

- □ knowledge about formal languages required (2nd semester)
  $\rightarrow$ easy: YES
- □ seven rules for the complete PLY grammar:
  $\rightarrow$ simple: YES

But it you will get some bonus...

Bonus

Random Access PLY

Why PLY?

Why PLY?
easy and simple?
easy and simple?
simplify
Qi
easy and simple?
new grammar
easy and simple?
▷ Bonus

```
#define LITTLE_ENDIAN

typedef struct {
    float32 x;
    float32 y;
    float32 z;
} vertex, vertices[8];

typedef struct {
    uint32 indices[3];
} face, faces[6];
```

The file header is now valid C code!