D:\tmp\gs_tutorial_code\michel_fodje_cpp_soln_gms_tutorial\small_matrix.hpp

```
#ifndef tMatrix_H
#define tMatrix_H

#include <ostream>
#include <iomanip>
#include <limits>


namespace Math {

template <class T>
inline T sign(T a, T b) {
 return ((b) >= 0.0 ? fabs(a) : -fabs(a));
}

template <class T>
inline T sqr(T a) {
 return (a * a);
}

template <class T>
inline void swap(T& a,T& b) {
        T y = a;
        a = b;
        b = y;
}

template <int order>
class tMatrix {
        protected:
                float mVal[order][order];
                int pos; // for operator , use only
        public:
       static const tMatrix<order> identity() {
          tMatrix<order> a;
        for ( int i = 0; i < order; ++i)
                for ( int j = 0; j < order; ++j)
                                        a(i,j) = (i==j) ? 1 : 0;
          return a;
       }

                tMatrix<order>(bool initialize=true) {
                        if (initialize) reset();
                }
                float& operator()( int i, int j) { return mVal[i][j]; }
                const float& operator()( int i, int j) const { return
mVal[i][j]; }
                tMatrix<order>& operator=(float d) {
                        int i = pos / order;
                        int j = pos % order;
                        mVal[i][j] = d;
                        pos++;
                        return *this;
                }

                const tMatrix<order> transpose(){
                        tMatrix<order> b;
        for ( int i = 0; i < order; ++i)
                for ( int j = i; j < order; ++j) {
                        b(j,i)=mVal[i][j];
                        b(i,j)=mVal[j][i];
                                }
                        return b;
                }

                void reset() {
           for ( int i = 0; i < order; ++i)
```

```
                for ( int j = 0; j < order; ++j)
                                        (*this)(i,j) = 0.0;
                  pos = 0;
                }

};

template <int order>
class tVector {
        public:
                int mVal[order];
                int pos; // for operator , use only
        float shift;
        bool centric;
        bool absent;

                tVector<order>(bool initialize=true) {
                        if (initialize) reset();
                        absent = false;
                        centric = false;
                }

                int& operator()( int i) { return mVal[i]; }
                const int& operator()( int i) const { return mVal[i]; }
                void reset() {
                for ( int i = 0; i < order; ++i)
                                (*this)(i) = 0;
                        pos = 0;
                }

        const tVector<order> operator-() const {
            tVector<order> tmp;
                for ( int i = 0; i < order; ++i)
                                tmp(i) = -mVal[i];
                        return tmp;
        }


};

template <int order>
std::ostream& operator<<(std::ostream& os, const tMatrix<order>& m) {
   using namespace std;
   os<<setiosflags(ios::fixed);
   os<<setprecision(3);
   os<<order<<" x "<<order<<"=\n";
   for(int i=0;i<order;++i) {
      os<<"[";
      for(int j=0;j<order;++j)
         os<<setw(8)<<m(i,j)<<",";
      os<<"]\n";
   }
        return os;
}

template <int order>
std::ostream& operator<<(std::ostream& os, const tVector<order>& m) {
   using namespace std;
   os<<setiosflags(ios::fixed);
   os<<setprecision(3);
   for(int j=0;j<order;++j){
        os<<setw(5)<<m(j);
   }
   return os;
}

template <int order>
```

```
const tVector<order> operator * (const tMatrix<order>& lhs,const
tVector<order>& rhs) {
        tVector<order> tmp;
        for ( int i = 0; i < order; ++i) {
                tmp(i) = 0;
                for ( int j = 0; j < order; ++j)
                        tmp(i) += static_cast<int>( lhs(i,j) * rhs(j) );
        }
        return tmp;
}

bool operator < (const tVector<3>& lhs, const tVector<3>& rhs) {
    if (lhs(0) < rhs(0)) {
        return true;
    } else if (lhs(0) == rhs(0)) {
        if (lhs(1) < rhs(1)) {
            return true;
        } else if (lhs(1) == rhs(1)) {
            if (lhs(2) < rhs(2)) {
                return true;
            } else return false;
        } else return false;
    } else return false;
}

bool operator == (const tVector<3>& lhs, const tVector<3>& rhs) {
    if (lhs(0) == rhs(0))
        if (lhs(1) == rhs(1))
            if (lhs(2) == rhs(2))
                return true;
    return false;
}

} // Math

#endif
```